Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних наук

ДИПЛОМНА РОБОТА

МАГІСТРА

на тему *Моделювання та адміністрування веб-орієнтованих систем*

Галузь знань *12 – Інформаційні технології*
Шифр і назва галузі знань

Спеціальність *122 – Комп'ютерні науки*
Шифр і назва спеціальності

Виконав: студент *2* курсу, група *КНм-20-1* _____ *І.І. Матвійчук*
Підпис Ініціали, прізвище

Керівник: *к.т.н., доцент кафедри КН* _____ *Р.О. Багрій*
Підпис Ініціали, прізвище

Нормоконтроль: *к.т.н., доцент кафедри КН* _____ *Р.О. Багрій*
Підпис Ініціали, прізвище

До захисту допускаю:
Зав. кафедри КНІТ, д.т.н., професор _____ *О.В. Бармак*
Підпис Ініціали, прізвище

3 грудня 2021 р.

Хмельницький 2021

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет *інформаційних технологій*
Кафедра *комп'ютерних наук*
Освітній ступінь *магістр*
Галузь знань *12 – Інформаційні технології*
Спеціальність *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ
Завідувач кафедри *комп'ютерних наук та інформаційних технологій*

_____
(підпис)
д.т.н., професор О.В. Бармак
«*1*» *вересня* *2021* року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ МАГІСТРА

1. Тема кваліфікаційної роботи магістра: «Моделювання та адміністрування веб-орієнтованих систем»

2. Завдання видано студенту *Матвійчуку Івану Ігоровичу*
(прізвище, ім'я, по батькові)

3. Керівник роботи *к.т.н., доцент Багрій Руслан Олександрович*
(прізвище, ім'я, по батькові)

4. Затверджені наказом університету від «*25*» *серпня* *2021* р. № *102*

5. Зміст пояснювальної записки (перелік задач) та вихідні дані:

Метою роботи - розробка технології моделювання та адміністрування web-орієнтованих систем, яка б надала можливість налаштування контенту для покращення швидкодії побудови та пошукової індексації сторінок.

Реферат

Кваліфікаційна робота магістра присвячена розробці інформаційної технології моделювання та адміністрування web-орієнтованих систем.

Актуальність теми. В теперішній час важко уявити собі людину яка ніколи не стикалася з певними інтернет ресурсами. Веб-системи супроводжують людину на кожному її кроці. Вони використовуються для вирішення великої кількості завдань, та мають великий спектр застосування. Для успішної веб-орієнтованої системи критично важливо мати потужну систему для контролю контенту цієї системи. Такі системи існують вже досить давно та називаються Системами керування контентом. Серед популярних систем керування вмістом слід відзначити Wordpress, Joomla, WooComerce. Ці системи мають хороший функціонал, але їх дуже важко персоналізувати, тобто додавати такий функціонал, який притаманний тільки для конкретного бізнесу. Створення кожної з таких систем потребує залучення великої кількості ресурсів та вирішення багатьох стратегічно важливих проблем. Серед проблем можна виділити такі:

- аналіз ринку та цільової аудиторії;
- побудова бізнес моделі роботи системи;
- створення дизайну майбутньої системи;
- вибір технологій розробки;
- розробка програмного продукту;
- запровадження продукту;
- підтримка та оновлення;

На противагу цим великим та важким системам було розроблено невелику систему, яка буде мати великі можливості для розширення та доповнення її новими функціями. Така система повинна містити функціонал для створення нових сторінок, роботи з навігацією на сайті, роботи з даними необхідними для пошукової оптимізації сайту. Окрім якісного контенту сайт повинен мати високу швидкість завантаження, чого досить важко досягнути для повністю динамічного контенту.

Мета і задачі роботи. Метою магістерської роботи є розробка технології моделювання та адміністрування web-орієнтованих системи, яка б надала можливість налаштування контенту для покращення швидкодії побудови та пошукової індексації сторінок.

Для досягнення поставленої мети були визначені такі задачі:

– дослідили види та основні принципи створення web-орієнтованих систем;

– розробити метод побудови гіпертекстової розмітки на сервері;

– дослідити фактори та способи оптимізації, що впливають на пошукову індексацію сторінок сайту;

– спроектувати та розробити технологію моделювання та адміністрування web-орієнтованої системи з можливістю керування процесом побудови сторінки на сервері та налаштуванням мета-інформації, що необхідна для пошукової індексації.

Об'єкт дослідження – процес моделювання веб-орієнтованих систем.

Предмет дослідження – моделі, методи, підходи та засоби для моделювання та адміністрування web-орієнтованих систем.

Наукова новизна одержаних результатів. В результаті проведеної роботи були отримані такі результати:

– вдосконалено алгоритм рендерингу динамічного контенту веб-сторінки, що дає можливість підвищити швидкість роботи веб-орієнтованої системи;

– вдосконалено метод пошукової оптимізація веб-сайту, що за допомогою визначеного набору інструментів дозволив покращити показники індексації веб-орієнтованої системи;

– розроблено технологію моделювання та адміністрування web-орієнтованої системи з можливістю керування процесом побудови сторінки на сервері та налаштуванням мета-інформації, що необхідна для пошукової індексації.

Практичне значення одержаних результатів. На основі розробленої технології моделювання та адміністрування веб-систем створено програмну реалізацію у вигляді панелі адміністратора з можливістю керування процесом побудови сторінки та налаштуванням мета-інформації, що дає можливість оптимізувати процес пошукової індексації.

Апробація результатів дипломної роботи магістра та публікації:

Основні наукові та практичні результати доповідалися на конференціях:

доповідь на тему «Моделювання web-орієнтованих систем» на XIII всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2021» 15 листопада 2021 р., м. Хмельницький, Україна.

Структура та обсяг роботи. Дипломна робота магістра складається з завдання, реферату, змісту, переліку скорочень, вступу, 4 розділів, висновків, переліку посилань із 30 найменувань. Загальний обсяг дипломної роботи магістра становить 102 сторінок, з них 80 сторінок основного тексту та 22 сторінок додатків. У роботі наведено 54 рисунки.

Ключові слова: панель адміністратора, веб-орієнтована система, пошукова оптимізація.

Зміст

Перелік скорочень

Скорочення, термін, позначення	**Пояснення**
ОС	Операційна система
ПЗ	Програмне забезпечення
ІС	Інформаційна система
ПА	Панель адміністратора
СКВ	Система керування вмістом

Вступ

Кваліфікаційна робота магістра присвячена розробці інформаційної технології моделювання та адміністрування web-орієнтованих систем.

Актуальність теми. В теперішній час важко уявити собі людину яка ніколи не стикалася з певними інтернет ресурсами. Веб-системи супроводжують людину на кожному її кроці. Вони використовуються для вирішення великої кількості завдань, та мають великий спектр застосування. Для успішної веб-орієнтованої системи критично важливо мати потужну систему для контролю контенту цієї системи. Такі системи існують вже досить давно та називаються Системами керування контентом. Серед популярних систем керування вмістом слід відзначити Wordpress, Joomla, WooComerce. Ці системи мають хороший функціонал, але їх дуже важко персоналізувати, тобто додавати такий функціонал, який притаманний тільки для конкретного бізнесу. Створення кожної з таких систем потребує залучення великої кількості ресурсів та вирішення багатьох стратегічно важливих проблем. Серед проблем можна виділити такі:

- аналіз ринку та цільової аудиторії;
- побудова бізнес моделі роботи системи;
- створення дизайну майбутньої системи;
- вибір технологій розробки;
- розробка програмного продукту;
- запровадження продукту;
- підтримка та оновлення;

На противагу цим великим та важким системам було розроблено невелику систему, яка буде мати великі можливості для розширення та доповнення її новими функціями. Така система повинна містити функціонал для створення нових сторінок, роботи з навігацією на сайті, роботи з даними необхідними для пошукової оптимізації сайту. Окрім якісного контенту сайт повинен мати високу

швидкість завантаження, чого досить важко досягнути для повністю динамічного контенту.

Мета і задачі роботи. Метою магістерської роботи є розробка технології моделювання та адміністрування web-орієнтованих системи, яка б надала можливість налаштування контенту для покращення швидкодії побудови та пошукової індексації сторінок.

Для досягнення поставленої мети були визначені такі задачі:

– дослідили види та основні принципи створення web-орієнтованих систем;

– розробити метод побудови гіпертекстової розмітки на сервері;

– дослідити фактори та способи оптимізації, що впливають на пошукову індексацію сторінок сайту;

– спроектувати та розробити технологію моделювання та адміністрування web-орієнтованої системи з можливістю керування процесом побудови сторінки на сервері та налаштуванням мета-інформації, що необхідна для пошукової індексації.

Об’єкт дослідження – процес моделювання веб-орієнтованих систем.

Предмет дослідження – моделі, методи, підходи та засоби для моделювання та адміністрування web-орієнтованих систем.

Наукова новизна одержаних результатів. В результаті проведеної роботи були отримані такі результати:

– вдосконалено алгоритм рендерингу динамічного контенту веб-сторінки, що дає можливість підвищити швидкість роботи веб-орієнтованої системи;

– вдосконалено метод пошукової оптимізація веб-сайту, що за допомогою визначеного набору інструментів дозволив покращити показники індексації веб-орієнтованої системи;

– розроблено технологію моделювання та адміністрування web-орієнтованої системи з можливістю керування процесом побудови сторінки на

сервері та налаштуванням мета-інформації, що необхідна для пошукової індексації.

Практичне значення одержаних результатів. На основі розробленої технології моделювання та адміністрування веб-систем створено програмну реалізацію у вигляді панелі адміністратора з можливістю керування процесом побудови сторінки та налаштуванням мета-інформації, що дає можливість оптимізувати процес пошукової індексації.

Апробація результатів дипломної роботи магістра та публікації:

Основні наукові та практичні результати доповідалися на конференціях:

доповідь на тему «Моделювання web-орієнтованих систем» на XIII всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2021» 15 листопада 2021 р., м. Хмельницький, Україна.

Структура та обсяг роботи. Дипломна робота магістра складається з завдання, реферату, змісту, переліку скорочень, вступу, 4 розділів, висновків, переліку посилань із 30 найменувань. Загальний обсяг дипломної роботи магістра становить 102 сторінок, з них 80 сторінок основного тексту та 22 сторінок додатків. У роботі наведено 54 рисунків.

Ключові слова: панель адміністратора, веб-орієнтована система, пошукова оптимізація.

Розділ 1
Аналіз сучасного стану систем моделювання web-орієнтованих систем.

1.1 Аналіз предметної області

Використання web-систем в наш вже важко назвати новинкою, більшість людей зараз активно користується декількома або навіть десятком таких систем. Ринок активно розвивається та розширює свої горизонти. Кожен бізнес для нормального існування потребує застосування інформаційних систем. Такі системи можуть виконувати велику кількість різноманітних функцій, від управління внутрішніми процесами компанії до рекламування бренду компанії серед користувачів мережі інтернет. Програмне забезпечення для керування внутрішніми процесами є досить унікальним і зазвичай розробляється під конкретні завдання і для конкретної компанії. Для рекламних цілей бізнес зазвичай користується декількома ресурсами одночасно, одним з яких є власний сайт.

Сайти компаній можна умовно розділити на 3 категорії: прості сайти візитки, повноцінні багатосторінкові сайти, інтернет застосунки. Сайт-візитка зазвичай являє собою односторінковий сайт з короткою інформацією та формою для зв’язку. Цей варіант чудово підходить для малого бізнесу. Для середнього та великого бізнесу краще підходить розробка повноцінного сайту, який буде з складатись із багатьох сторінок(головна, про нас, блог, новини, вакансії) та буде мати панель адміністратора для керування контентом та іншими функціями. Третя група поєднує в собі як рекламні, так і функціональні цілі. Тобто цей програмний засіб і є послугою яку пропонує бізнес. До такого програмного забезпечення можна віднести соціальні мережі, сайти для оголошень, інтернет-магазини. Для розробки кожного цих типів програмного забезпечення потрібно використовувати різні підходи. Для розробки сайту-візитки доцільно скористатись онлайн конструктором, або звернутись до веб-студії для розробки

унікального дизайну. Для розробки повноцінних сайтів можна скористатись системою керування вмістом, такі системи є досить поширеними, але не гарантують належної якості продукту, також можна залучити команду розробників, які створять якісний продукт, але на це піде більше часу та коштів. Для третього типу програмного забезпечення потрібно в будь якому випадку залучити велику команду розробників та великі фінансові ресурси.

Окрім сайтів часто можуть використовуватись і інші типи web-орієнтованих систем. Веб-орієнтована система — це будь який програмний продукт для коректної роботи якого необхідне з’єднання з мережею інтернет. До веб-орієнтованих систем можна віднести: сайти, мобільні додатки, настільні додатки.

Більшість веб-орієнтованих систем[1] мають приховану від звичайних користувачів частину, що називається панель адміністратора. Панель адміністратора[2] - це набір набір інструментів, що надає можливість повного керування системою. Функціонал панелі адміністратора є унікальним для кожної для кожної веб-орієнтованої системи. Основною функцією панелі адміністратора є прискорення часу реагування власників системи на дії користувачів або якісь зміни. Панель адміністратора зазвичай складається із двох основних частин: розділів для роботи із сутностями із бази даних та аналітики. Розділи для роботи із сутностями із бази даних надають функціонал для виконання чотирьох основних операцій над даними: читання, редагування, створення, видалення. Прикладом такого розділу може бути розділ із заявками на вакансії, що розміщені на сайті компанії. Після того як користувач відправляє заповнену форму з інформацією про себе та прикріплює резюме, у базі даних створюється відповідна сутність. Після створення такої сутності адміністратор бачить її у відповідному розділі та приймає рішення відповідно до вимог компанії. Аналітична частина панелі адміністратора носить в основному інформативний характер та надає можливість адміністратору побачити які дії виконуються користувачами в рамках

цієї системи. Наявність такої інформації дає повне розуміння того, що користується популярністю в системі та в який бік слід цю систему розвивати.

Важливою складовою кожної веб-орієнтованої системи є її оптимізованість. Оптимізованість[3] — це кількість часу, за який веб-сторінка повністю завантажується. Швидкість завантаження розраховується з моменту, коли користувач натискає на посилання, до моменту, коли система повністю завантажується, включаючи зображення, відео тощо. Швидкість завантаження системи має дуже велике значення, що підтверджує наступна статистика:

– 1 з 4 відвідувачів залишає веб-орієнтовану систему, завантаження якої займає більше 4 секунд;

– 46 відсотків користувачів не повертаються на неоптимізовані системи;

– Сайти електронної комерції з найвищим коефіцієнтом конверсії повністю завантажуються менш ніж за 2 секунди;

Крім цього швидкість завантаження є важливим фактором рейтингу Google. Цей факт було підтверджено у 2018 році, коли в компанії Google швидкість завантаження сторінки зробили одним із найважливіших факторів рейтингу для пошуку з мобільних пристроїв. Збільшення конверсій є кінцевою метою кожного бізнесу. Конверсія[4] — пропорція або залежність числа покупок від кількості відвідувачів. Швидкість завантаження сторінки є вирішальним фактором для прийняття рішення про покупку. Рекомендований час завантаження системи компанією Google становить менше 2 секунд. Для визначення швидкості завантаження системи існують спеціальні інструменти, серед який слід відзначити наступні:

– PageSpeed Insights[5]: вказує, наскільки ефективна ваша сторінка, і пропонує оптимізацію;

– Lighthouse[6]: чудовий інструмент для перевірки продуктивності, доступності та іншого веб-сторінок;

Швидкість завантаження може залежати від великої кількості факторів. Деякі приклади включають типи вашого веб-хостингу та сервера, код і бази даних,

які використовує ваш веб-сайт, вибір елементів дизайну, кількість і розмір зображень, відео та файлів на кожній сторінці, версії браузера, поведінку користувачів тощо.

Основний обсяг контенту сайту займають медіафайли. Для вирішення цієї проблеми можна скористатися програмним забезпеченням, яке зменшить обсяг пам’яті, що займають медіа-файли без втрати якості. Іншим способом вирішення такої проблеми є оптимізація таких файлів під час завантаження їх на сервер. Для цього в кожній мові програмування існують готові модулі. Після оптимізації розмір кожного файлу може зменшитись у декілька разів. Але крім зменшення розміру файлів слід скористатися можливістю асинхронного завантаження. Асинхронне завантаження[7] — це призупинення завантаження медіа-контенту до моменту, коли не буде завантажено всю гіпертекстову розмітку сторінки. Для цього в гіпертекстовій розмітці існує спеціальний атрибут під назвою loading. Ще одним способом оптимізації медіа-контенту є використання вбудованих можливостей серверу.

Наступним кроком оптимізації веб-орієнтованої системи є зменшення обсягу коду, який транспортується мережею. Зазвичай для цього використовуються мініфікація цього коду. Мініфікація[8] — це процес з'єднання великої кількості файлів в один із видаленням форматування.

```
2  MIT License (c) copyright 2011-2013 original author or authors */
3  window._bvt||(window._bvt=!0,function(r,D){function jb(a){function d(a,b
4  a[da]("data-node-uid"))&&delete fa[uid]}function l(a){try{return"true"==
5  function(b){if(!b[y]||b[y]&&!b[y][y]){var d=b.cloneNode(!0);f.$&&f.clone
6  a.length;for(var b=0;b<a.length;b++)this[b]=a[b]}}function P(a){return"s
7  B=a,V=B.document,z=V.documentElement,y="parentNode",G=null,q=/^checked|v
8  x=/^checked|selected$/,v=/msie/i.test(navigator.userAgent),fa={},Va=0,Ja
9  "MozTransform","OTransform","msTransform","Transform"],k;for(k=0;k<f.len
10 b){var c=100;try{c=a.filters["DXImageTransform.Microsoft.Alpha"].opacity
```

Рисунок 1.1 – Приклад мініфікованого коду

Іншою важливою складовою кожної веб-орієнтованої системи є її пошукова оптимізованість. Пошукова оптимізація веб-орієнтованої системи (англ search engine optimization) [9] — це перелік заходів та дій, що включають в себе редагування наповнення(контенту), структури, метою якого є підняття рейтингу

системи в результатах пошуку за певними запитами користувачів. Чим вище позиція сайту в результатах пошуку, тим більша ймовірність, що відвідувач перейде на нього з пошукових систем, оскільки люди зазвичай йдуть за першими посиланнями.

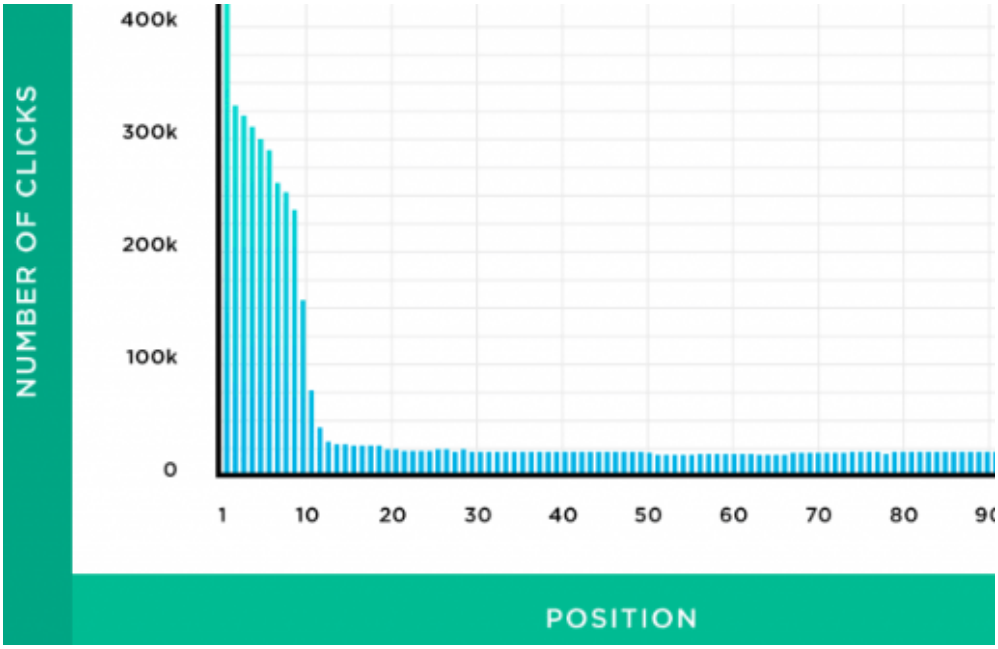


Рисунок 1.2 – Приклад залежності рейтингу сторінки в пошукових запитах з її відвідуваністю.

Пошукова оптимізація з’явилася із появою перших пошукових систем в середині 90-х років 20-го століття. Перші методи пошукової оптимізації передбачали редагування контенту та мета-тегів. Внаслідок таких заходів високі місця у пошуку стали отримувати сайти, котрі не містили корисного змістового навантаження, а лише популярні пошукові запити. У той час пошуковики надавали великого значення тексту на сторінці та іншим внутрішнім чинникам, якими власники сайтів могли легко маніпулювати. Це призвело до того, що у видачі багатьох пошуковиків перші декілька сторінок зайняли так звані «сайти-смітники», що різко знизило якість роботи пошуковиків і спричинило занепад багатьох із них. З появою технології PageRank більшої уваги стали приділяти зовнішнім чинникам, що допомогло Google вийти в лідери пошуку у світовому масштабі, ускладнивши оптимізацію за допомогою одного лише тексту на сайті.

Одними із основних термінів пошукової оптимізації є релевантність та ранжування. Релевантність(англ Relevance)[10] у інформаційній науці і інформаційному пошуку означає ступінь відповідності знайденого документа або набору документів інформаційним потребам користувача. Ранжування[11] — це процес розміщення даних у певному порядку за ступенем важливості, значущості. У процесі ранжування сайту пошуковими системами беруться до уваги більш ніж 200 факторів.

Основними факторами, що впливають на видачу результату в пошукових системах є:

– внутрішня оптимізація сторінки;

– технічна оптимізація сайту (використовувана CMS, чистота коду, швидкість завантаження сайту, мобільна версія тощо);

– якісні зовнішні посилання на сайт;

– вік сайту;

– контент для людей (ключових слів може в статті не бути, але вона буде на першій позиції через релевантність і цінність);

– наявність ключових слів в адресі сайту

– До факторів, що знижують рейтинг сайту, належать:

– завелика кількість ключових слів у контенті;

– технології, які пошукові машини розглядають як спам;

Для пошукової оптимізації веб-орієнтованої системи існує декілька основних типів заходів(Зовнішня пошукова оптимізація, біла оптимізація[12], сіра оптимізація, чорна оптимізація[13]).

Методи зовнішньої пошукової оптимізації:

– Обмін посиланнями. Існують декілька способів обміну — прямий, взаємний, односторонній (покупка посилань);

– Розміщення статей;

– Соціальні мережі;

– Прес-релізи;

– Активна діяльність на тематичних блогах;

– Створення та ведення блогів.

Біла оптимізація — оптимізаторська робота над ресурсом без застосування офіційно заборонених кожною пошуковою системою методів розкручування ресурсу — без впливу на пошукові алгоритми сайтів. Вона включає в себе роботу над самим сайтом, тобто над його внутрішньою навігацією і вмістом, і роботу з зовнішнім середовищем сайту, тобто просуванням цього сайту шляхом оглядів, прес-релізів, реєстрації в соціальних закладках, партнерських програм тощо із зазначенням посилань на сайт. Слід зазначити, що навіть якщо який-небудь метод оптимізації не є офіційно забороненим, це не означає, що його можна застосовувати.

Сіра оптимізація. До сірої пошукової оптимізації можна віднести додавання великої кількості ключових слів в текст сторінки, часто з втратою прочитності для людини. Проте в деяких випадках при використанні сірої оптимізації допускається помилка під назвою переоптимізація, що призводить до переходу з сірої оптимізації в чорну. При цьому завдання SEO-копірайтера — написати оригінальний текст таким чином, щоб подібна оптимізація була якомога менш помітна «живому» читачеві. Сіра оптимізація відрізняється від чорної тим, що вона офіційно не заборонена, але її використання все одно може бути розцінене як неприродне завищення популярності сайту. Деякі пошукові системи, наприклад, Google, можуть тимчасово або назавжди заблокувати такий сайт. Тобто, остаточне рішення про те, чи є методи просування законними чи ні, приймає фахівець — модератор пошукової системи, а не програма.

Чорна оптимізація. До чорної оптимізації відносяться всі методи, які суперечать правилам пошукових систем. Серед них можна виділити наступні: використання дорвеїв (сторінок і ресурсів, створених спеціально для роботів пошукових систем, найчастіше з великою кількістю ключових слів на сторінці), прийом під назвою клоакінг (користувачеві віддається одна сторінка, що легко

читається, а пошуковому роботу — інша, оптимізована під які-небудь запити), використання прихованого тексту на сторінках сайту.

Отже можна зробити висновок, що ступінь пошукової оптимізації сторінки залежить від двох основних критеріїв: якість контенту представленого на веб сторінці, а також рівень технічної реалізації цієї веб-сторінки. До контенту, що впливає на пошукову оптимізацію відноситься мета-інформація та текст, що відображається в тілі сторінки. У розробленій панелі адміністратора пропонується повне керування контентом як самої сторінки, так і її мета-інформації. Хоча система буде надавати повний контроль над мета-інформацією та контентом сайту, вона не може гарантувати його якість, ця відповідальність лягає на плечі контент-менеджера.

1.2 Аналіз видів веб-орієнтованих систем

Як було сказано вище, web-орієнтована система — це система для роботи якої потрібне постійне з’єднання з мережею інтернет. Існує кілька найбільш поширених видів таких систем: сайти, мобільні застосунки та застосунки для настільних комп’ютерів.

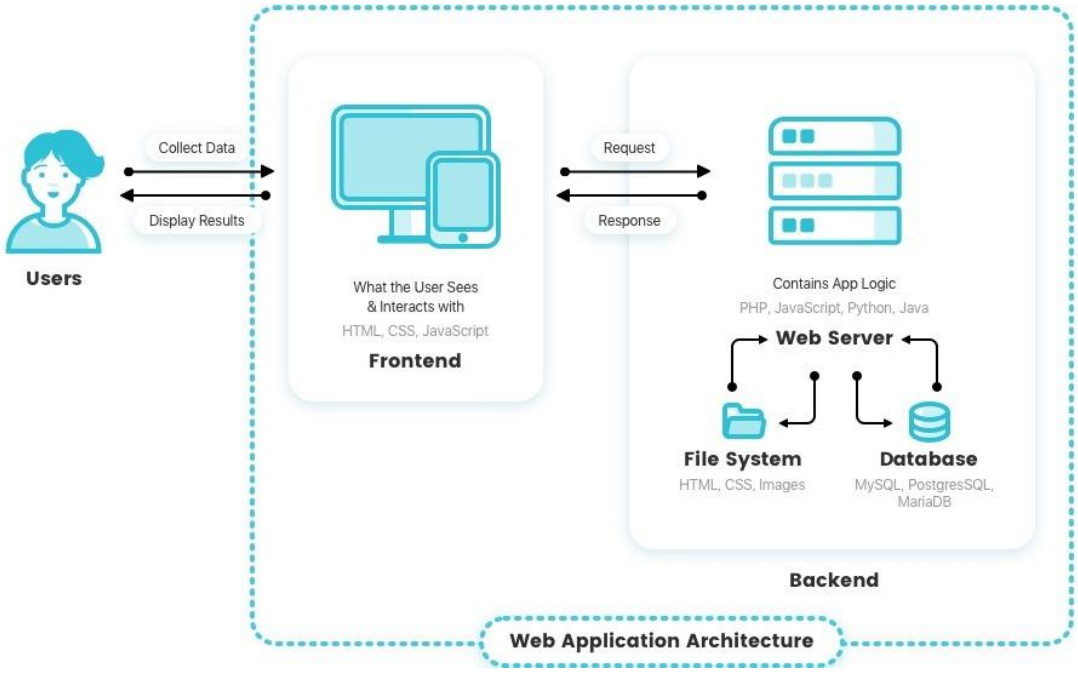


Рисунок 1.3 – Схема роботи web-орієнтованої системи

Для розробки web-орієнтованих систем використовуються різні типи технологій та засобів:

– Операційні системи[14]

– Веб-сервери[15]

– бази даних та системи керування базами даних[16]

– Фреймворки для серверної частини[17]

Операційна система(Linux, Windows, MacOS) — програмне забезпечення для керування ресурсами комп'ютера, організовують взаємодію з користувачем.

Веб-сервер(Apache, nginx) — програмне забезпечення, яке використовується для опрацювання HTTP-запитів, запуск певного коду системи, та відправку результатів клієнту.

База даних та СКБД(MySQL, MongoDB, MsSQL, PostgreSQL) — сукупність даних та зв'язків між ними. Система керування базами даних — система, що надає можливість виконувати базові операції над даними(створення, відображення, редагування, видалення).

Фреймворки — системи, які являють собою каркас, що визначає структуру та базовий функціонал проекту. У фреймворку можуть бути готові набори модулів, бібліотек чи віджетів.

Правильний вибір технологій розробки є однією з найголовніших проблем з якою можуть зіткнутися розробники. Вибір технології дуже сильно впливає на швидкість та якість розробки системи. Неправильний вибір технології може поставити хрест на цілому проекті. Для вибору технологій необхідно володіти неабияким досвідом в програмуванні та моделюванні майбутнього продукту. Незважаючи на те що кожна система унікальна і потребує індивідуального підходу, для спрощення старту розробки можна виділити найбільш універсальні технології та створити на їх основі шаблон веб-орієнтованої системи, а саме шаблон панелі адміністратора. Іншою важливою складовою успішної web-орієнтованої системи є правильний підхід до контенту розміщеного всередині цієї

системи. Правильність заповнення системи даними впливає на індексування системи в пошукових запитах.

Веб Сайт[18] — це певний набір сторінок, які об’єднанні однією темою та доступні користувачам у мережі інтернет за доменним ім’ям. Це дуже загальне визначення поняття веб сайт, яке не відображає сучасних підходів до їх створення.

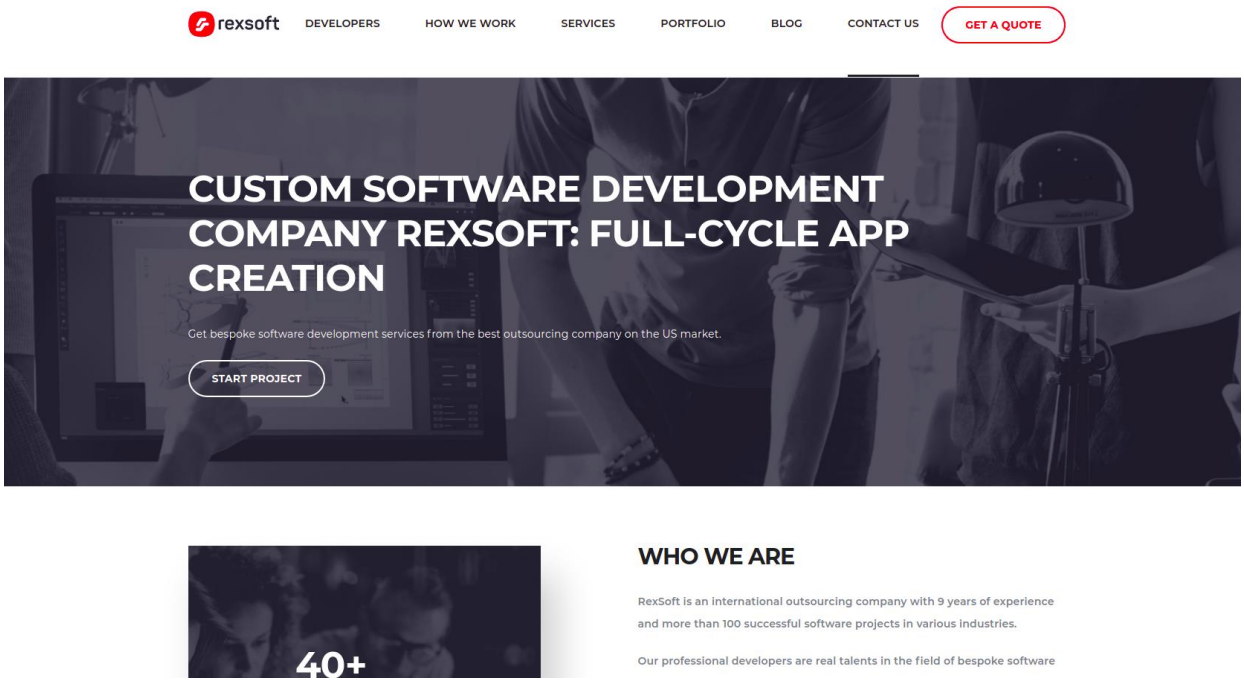


Рисунок 1.4 – Приклад web-орієнтованої системи — web-сайт компанії-розробника програмного забезпечення

Зазвичай Веб Сайти працюють за такою схемою[19]:

– Користувач відкриває браузер на своєму пристрої та вводить адресу сайту в пошуковому рядку;

– Запит користувача обробляється системою доменних імен иа перенаправляється на ip-адресу серверу;

– Сервер отримує запит користувача, обробляє дані, та повертає результат у вигляді html-розмітки;

– Браузер користувача отримує відповідь від сервера та відображає отриманий результат.

За такою схемою веб працював протягом багатьох років, але починаючи з 2013 року на ринку починають з’являтися нові технології, які кардинально змінюють принцип роботи веб сайту. Такими технологіями є React, Vue та Angular.

Ці технології вводять таке поняття як односторінковий веб сайт, де контент сторінки динамічно змінюється можливостями використаної технології. Такий програмний засіб складається із двох основних частин: клієнтський застосунок та серверний застосунок. Він працює за такою схемою:

– Користувач відкриває браузер на своєму пристрої та вводить адресу сайту в пошуковому рядку;

– Запит користувача обробляється системою доменних імен иа перенаправляється на ip-адресу клієнту;

– Клієнт отримує запит користувача та відправляє йому весь html, css та javascript код.

– Після того як браузер отримує код від клієнта, він починає його виконання;

– В ході виконання браузер робить запити на сервер для отримання динамічної інформації;

– Після отримання якої він з’єднує її з відповідною html-розміткою та відображає на екрані.

Кожен з цих підходів до роботи системи має свою назву. Ці назви вони отримали у відповідність до типу поєднання даних з їх відображенням, тобто розміткою. Перший тип носить назву SSR, що означає рендеринг на стороні серверу, другий тип називається SPA, що означає односторінковий застосунок.

Кожен з цих типів веб сайтів має свої переваги та недоліки. Серед переваг рендерингу на стороні серверу слід відзначити хорошу індексацію у пошукових системах та швидкість розробки, а до недоліків варто віднести необхідність перерендингу сторінки при кожному переході, важкість роботи з динамічними інтерфейсами. До переваг рендерингу на стороні клієнта слід віднести швидкість

роботи системи, адже перерендеринг буде відбуватись тільки у тому місці де змінилися дані, зручність роботи з динамічними інтерфейсами, наявність серверного інтерфейсу, який може використовуватись для роботи з іншими клієнтами, до недоліків слід віднести погану індексацію пошуковими системами, важкість розробки та швидкість першого завантаження.

Іншим видом web-орієнтованої системи є мобільний застосунок. Мобільний застосунок або додаток — програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях. Багато мобільних застосунків встановлені на самому пристрої або можуть бути завантажені на нього з онлайн магазинів мобільних застосунків, таких як App Store, Google Play, Windows Phone Store та інших, безкоштовно або за плату. Він працює за схожим принципом із односторінковим вебсайтом. Єдина різниця полягає у зміні клієнту. Тепер клієнтом виступає мобільний додаток, який робить запити до серверу та відображає результати.

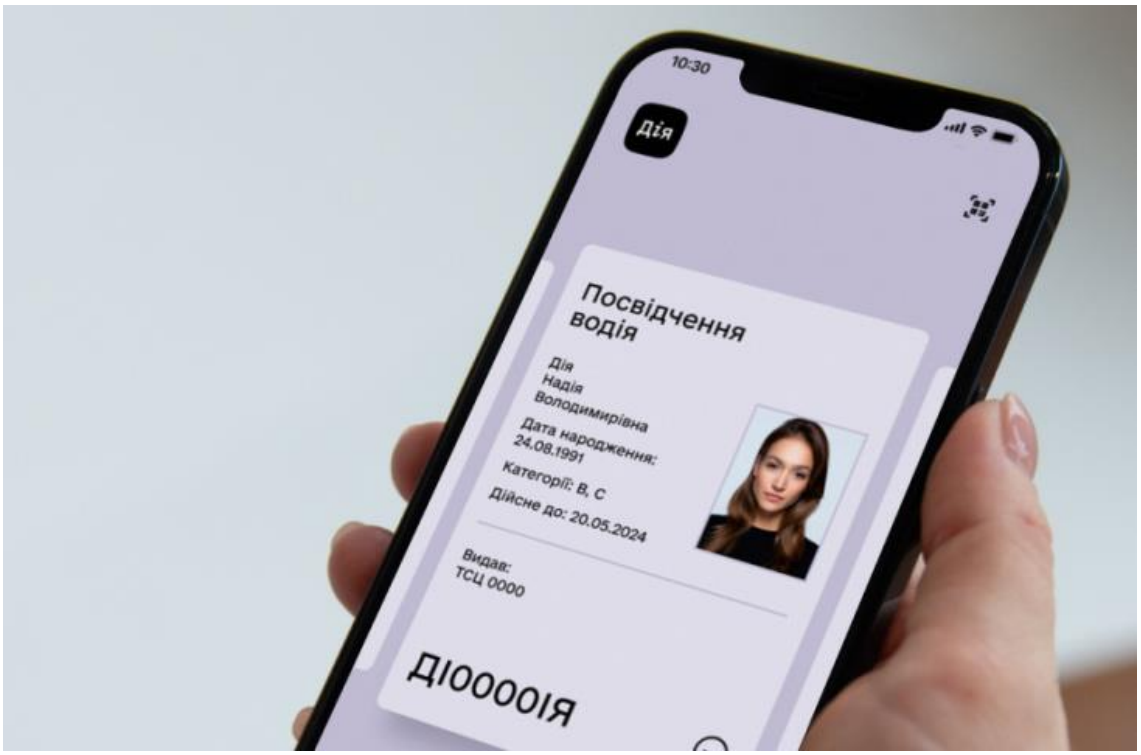


Рисунок 1.5 – Приклад web-орієнтованої системи — мобільний застосунок Дія

1.3 Аналіз існуючих систем керування вмістом

Система керування вмістом[20][21] — це програмне забезпечення, яке допомагає користувачам створювати, керувати та змінювати вміст веб-сайту без потреби в спеціальних технічних знаннях. Простішою мовою, система керування вмістом — це інструмент, який допомагає вам створити веб-сайт без необхідності писати весь код з нуля. Для розробки власної системи керування вмістом. було розглянуто найбільш популярні системи-аналоги. Такими системами є: WordPress, Joomla.

WordPress— система керування вмістом з відкритим кодом, яка через свою простоту в установленні та використанні широко застосовується для створення веб сайтів. Перша версія WordPress була створена в 2003 році Меттом Малленвегом і Майком Літтлом. Вона починалася як проста платформа, розроблена для людей, які хотіли створювати основні блоги та розміщувати їх в Інтернеті. Однак з часом він перетворився на гнучкий, потужний інструмент для створення практично будь-якого типу сайтів.

Переваги використання WordPress:

– WordPress – це повністю безкоштовна система управління контентом. Всі файли та база даних сайту знаходяться повністю під контролем власника. Власник може вибирати хостинг для сайту за власним бажанням і потребам.

– Наявність підтримки з боку розробників. Ядро системи часто оновлюється, додаються нові модулі, віджети, хукі, посилюється безпеку і швидкість роботи. У мережі є відповіді на практично будь-які питання, що можуть стосуватися цієї CMS.

– Тисячі плагінів. Один тільки репозиторій WordPress містить більше ніж 1000000 безкоштовних плагінів для вирішення різних типів завдань.

– Зручність і простота панелі адміністратора. Адмін-панель WordPress вкрай проста у використанні. Навіть люди без досвіду зможуть додавати нові розділи, сторінки, новини, зображення. Для складних сайтів — адмінка легко

адаптується під будь-які завдання за допомогою плагіна – “Advanced Custom Fields”.

– Висока швидкість роботи при правильній побудові шаблону, включеному кешуванні та правильній оптимізації зображень та контенту.

Недоліки використання WordPress:

– Необхідний постійний контроль за сайтом, за його безпекою та правильністю роботою. Установка плагінів для автоматизації роботи не знімають необхідності постійного контролю.

– Велика кількість неякісних плагінів, після установки яких можуть виникати конфлікти між ними. Через це швидкість завантаження сайту буде не високою та погіршить якість сайту в очах пошукових систем. А це у свою чергу дуже сильно погіршить позиції сайту в пошуковій видачі.

– WordPress це не сайт-билдер типу Wix, Shopify, Weebly або Jimdo. Тому, щоб зробити сайт з унікальною архітектурою і дизайном необхідно верстати все його сторінки з нуля і тільки після цього інтегрувати їх з CMS WordPress.

– Вразливість системи. Через популярність WordPress він є основною ціллю для хакерів і спамерів. Навіть якщо використовувати усі плагіни, що відповідають за безпеку системи, які пропонує WordPress, цього буде недостатньо для повного захисту та збереження системи.

– Погана оптимізація для пошукових систем. WordPress дозволяє лише обмежені функції пошувої оптимізації та не може гарантувати високі позицію у рейтингах.

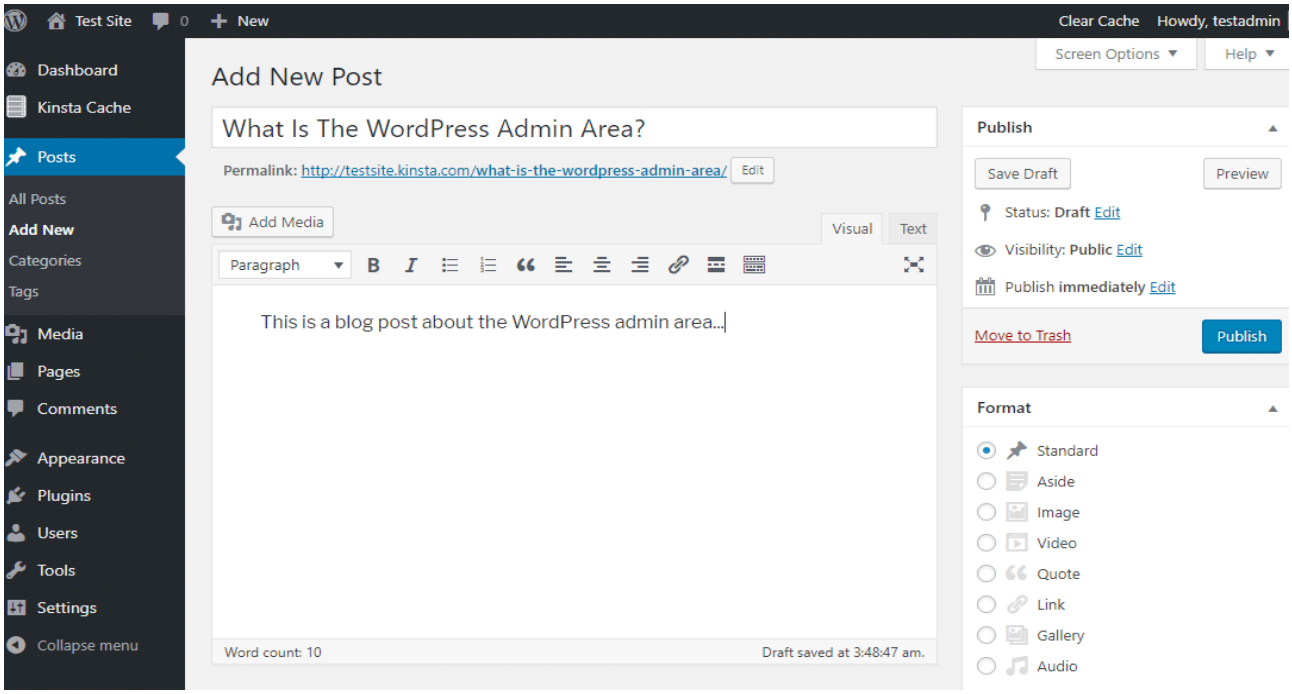


Рисунок 1.6 – Приклад адмін панелі сайту створеного з Wordpress

Joomla! — відкрита універсальна система керування вмістом для публікації інформації в інтернеті. Підходить для створення маленьких і великих корпоративних сайтів, інтернет порталів, онлайн-магазинів, сайтів спільнот і персональних сторінок. З особливостей Joomla можна відзначити: гнучкі інструменти управління обліковими записами, інтерфейс для управління медіа-файлами, підтримка створення багатомовних варіантів сторінок, система управління рекламними кампаніями, адресна книга користувачів, голосування, вбудований пошук, функції категоризації посилань і обліку кліків, WYSIWYG-редактор, система шаблонів, підтримка меню.

Переваги використання Joomla:

– Універсальність. Joomla — повноцінний інструмент для створення будь-якого сайту, попри відносну простоту, порівняно, наприклад, із Drupal.

– Зручність використання. Адмін-панель проста та зручна. Необхідні для налаштувань розділи та розширення легко знайти.

– Простота встановлення.

– Вбудована система кешування. Відкриті раніше вебсторінки система відправляє в кеш, що прискорює завантаження сторінок. Коштом цього немає зайвого навантаження на сервері, що покращує SEO-показники.

Недоліки використання Joomla:

– Багато розширень та шаблонів для Joomla платні.

– Немає технічної підтримки.

– Слабке місце цієї CMS — розширення, завантажені з сумнівного джерела, вони часто є причиною взламування.

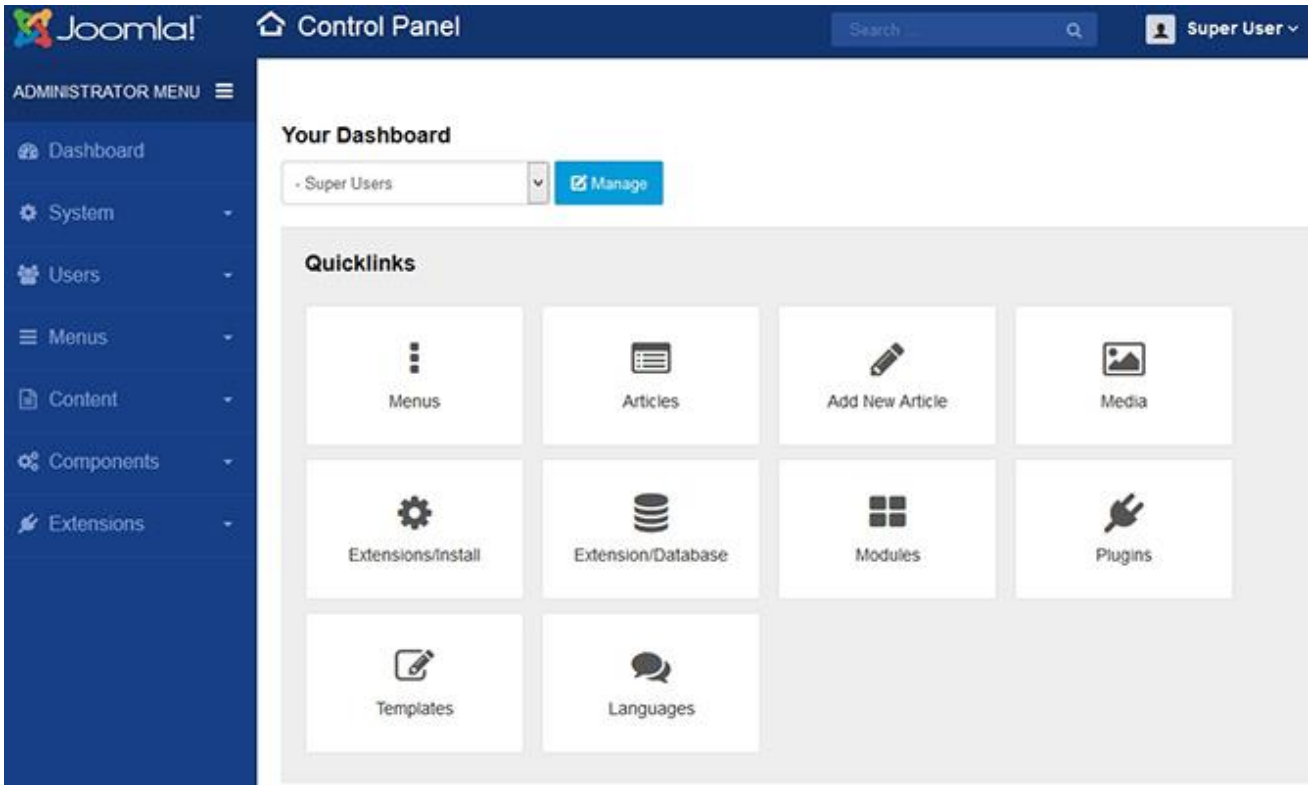


Рисунок 1.7 – Приклад адмін панелі сайту створеного з Joomla!

Розглянуті системи моделювання використовуються на багатьох веб сайтах, проте сайт написаний з використанням цих систем важко оновлювати, слідкувати за його безпекою та оптимізувати його для хорошого індексування в пошукових системах. Як альтернативу таким системам можна створити власну

систему моделювання, яка буде позбавлена багатьох недоліків існуючих систем та буде добре маштабуватись для задоволення потреб бізнесу.

1.4 Постановка задачі

Метою магістерської роботи є розробка технології моделювання та адміністрування web-орієнтованих систем, яка б надала можливість налаштування контенту, інформації необхідної для пошукової індексації сторінок та гарантувала швидкість роботи системи.

Для досягнення поставленої мети визначено наступні задачі:

– дослідити процес побудови гіпертекстової розмітки на сервері;

– дослідити способи оптимізації web-сайтів;

– дослідити, які фактори впливають на пошукову індексацію сторінки сайту;

– спроектувати та розробити технологію моделювання та адміністрування web-орієнтованої системи з можливістю керування процесом побудови сторінки на сервері та можливістю налаштування мета-інформації, що необхідна для пошукової індексації.

Висновки до розділу 1.

В першому розділі було визначено, що таке web-орієнтована система, та було розглянуто основні їх типи. Після проведеного аналізу було обрано найпоширеніший тип такої системи та розглянуто засоби для їх моделювання. Такі засобами називаються системами контролю вмісту. Такі системи є досить поширеними, але мають ряд недоліків. Як альтернативу цим засобам, було вирішено розробити власний засіб без притаманних таким системам недоліків.

Розділ 2

Дослідження факторів ефективності веб-орієнтованих систем.

2.1 Метод для побудови гіпертекстової розмітки на сервері

Одним із найважливіших факторів ефективності веб-орієнтованої системи є її оптимізованість. Основні проблеми, що можуть вплинути на показник оптимізованості описані у підрозділі 1.1. Для вирішення кожної з них під час побудови динамічної сторінки на сервері слід розглянути загальну схему роботи серверу та визначити, яким чином можна застосувати загальновідомі методи вирішення.

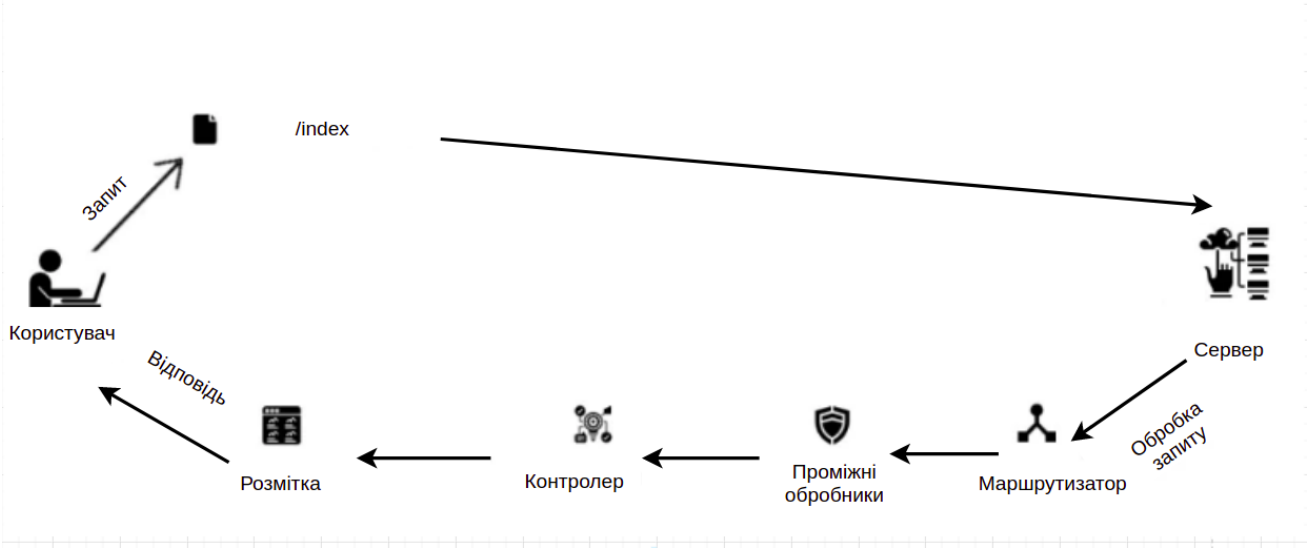


Рисунок 2.1 – Схема роботи веб-системи

Як зображено на рисунку 2.1, веб-система працює за таким алгоритмом:

– Користувач відкриває браузер та вводить адресу сайту в полі для пошуку;

– Введена користувачем адреса трансформується в ip-адресу сайту можливостями системи доменних імен;

– Сервер отримує запит від користувача та відправляє результат опрацювання цього запиту у вигляді гіпертекстової розмітки;

– Браузер користувача отримавши відповідь від серверу відображає отриманий результат на екрані.

Це загальна схема роботи веб сайтів і вона ніяким чином не відображає процеси, які відбуваються на сервері. Для того щоб зрозуміти як правильно побудувати систему контролю вмістом потрібно більш детально їх розглянути.

У таких випадках роботу серверу можна розділити на такі кроки:

– отримання вхідних даних;

– пошук логіки, що відповідає даному запиту;

– валідація вхідних даних;

– виконання бізнес-логіки та запитів до бази даних;

– підстановка динамічних даних у гіпертекстову розмітку;

– повернення результату до клієнту;

Така схема роботи є досить простою та чудово підходить для повністю сформованих гіпертекстових сторінок із деякими блоками з динамічним вмістом.

Проте така схема значно ускладнюється у випадку коли структура сторінки веб сайту не є заздалегідь визначеною, а формується з певних налаштувань.

На перший погляд може здатись, що у даній ситуації доцільно скористатись WYSIWYG-редактором, але це не так. Такі редактори доцільно використовувати для публікацій у блозі, де стилізація контенту переходить на другий план, але для побудови повноцінних сторінок він ніяк не підходить. З використанням такого редактору створення нової сторінки може перетворитися на процес, який здалека нагадує верстку, ще однією проблемою є те, що контент буде повністю статичним. Контент можна буде редагувати, але його ніяк не можна буде поєднати з даними із бази.

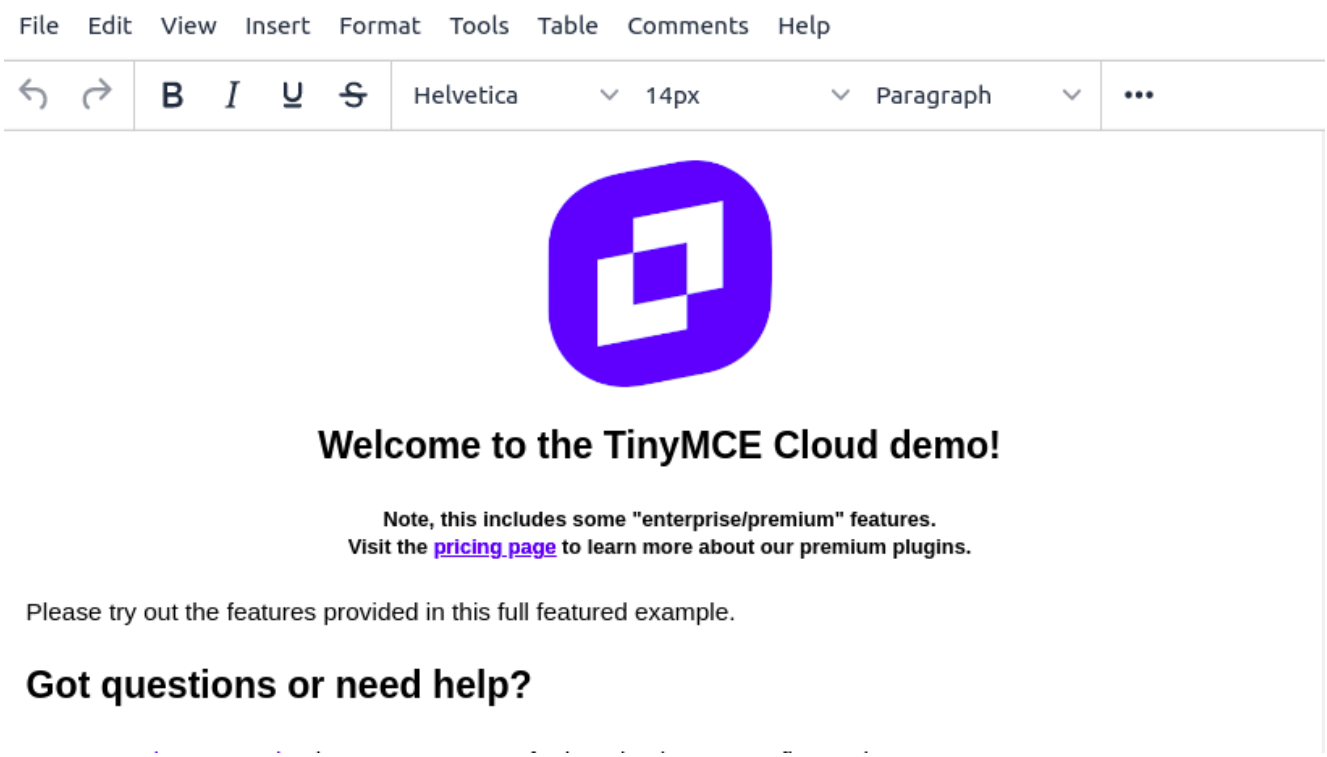


Рисунок 2.2 – Приклад WYSIWYG-редактором під назвою TinyMCE

Отже для створення ефективної веб-орієнтованої системи потрібно знайти інший шлях рендерингу динамічного контенту.

Основну ідею вирішення цієї проблеми було запозичено у фронтенд фреймворків мови програмування JavaScript. Ця ідея полягає в робитті гіпертекстової розмітки на блоки, що називаються компоненти. Ці компоненти заповнюються динамічними даними, та поєднуються у повноцінну веб сторінку.

Для реалізації поставленої задачі необхідно вдосконалити процес побудови гіпертекстової розмітки веб сторінки на сервері використовуючи усі відомі підходи до оптимізації.

Початком побудови сторінки є отримання інформації про блоки, що були використані під час моделювання цієї сторінки. Після одержання цієї інформації необхідно взяти інформацію по кожному блоку із конфігураційних файлів. У цих файлах описано запити до бази даних, що необхідні при рендерингу конкретного блоку, інформацію про наявність власних стилів, скриптів.

Після одержання цієї інформації необхідно паралельно виконати запити до бази даних для кожного блоку, після чого завантажити розмітку для блоків та підставити у відповідні місця динамічні дані. Отриману розмітку блоків потрібно поєднати в одне ціле та помістити у основний файл розмітки всередину тегу “ƅody”. Крім побудови розмітки контенту сторінки необхідно на основі отриманої інформації з конфігураційних файлів збудувати стилі, що повинні застосовуватися для сторінки, що будується. Для цього слід завантажити у пам’ять серверу стилі кожного компоненту, що використаний на сторінці. Потім необхідно поєднати їх у один. Під час поєднання слід перевіряти стилі на дублювання. Отриманий результат потрібно помістити у тег “style” основного файлу розмітки. Подібні дії потрібно виконати і у випадку побудови клієнтського коду. Єдиною відмінністю є те, що його слід помістити не у тег “style”, а у тег “script” .

Спираючись на все описане вище було розроблено метод рендерингу[22], тобто побудови повністю динамічної та оптимізованої веб сторінки(Рисунок 2.3), який складається з наступних кроків:

- отримати інформацію про використані компоненти на веб сторінці;
- завантажити гіпертекстову розмітку кожного компоненту;
- зробити запити до бази даних для отримання динамічного контенту;
- сформувати файл стилів, у відповідності до використаних компонентів;
- сформувати весь необхідний джаваскрипт код;
- надіслати сформований результат браузеру користувача;

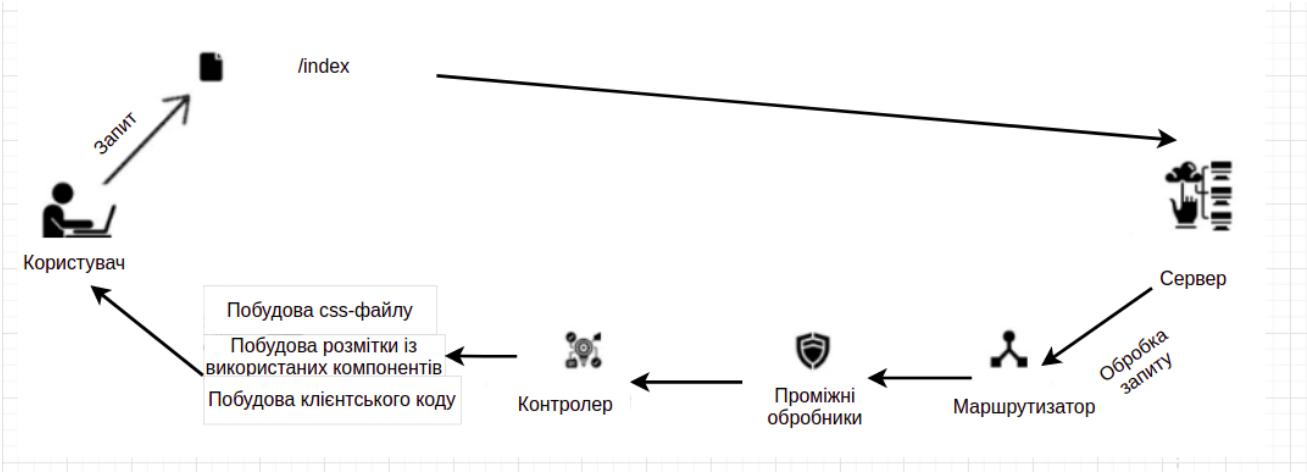


Рисунок 2.3 – Рендеринг оптимізованої веб сторінки

Реалізація цього підходу до побудови сторінки на сервері буде гарантувати швидкість роботи веб-орієнтованої системи, оскільки тільки необхідні дані будуть транспортуватися мережею інтернет.

2.2 Проектування методу роботи з пошуковою оптимізацією проектованої технології моделювання

Як було визначено у розділі 1.1 пошукова оптимізованість є важливою складовою успішності кожної веб-орієнтованої системи. Основними складовими, що впливають на пошукову опимізованість є контент системи та мета-інформація.

Для роботи з пошуковою оптимізацією у проектованій системі пропонується повний контроль над контентом сайту та його мета-інформацією.

Мета-інформація заповнюється в спеціальних тегах гіпертекстової розмітки веб сторінки. Основними мета-тегами[23] є title, description, keywords. Окрім основних тегів також є допоміжні серед яких слід виділити og-теги. Ці теги використовують технологію OpenGraph[24]. Open Graph був представлений Facebook у 2010 році. Він був розроблений для інтеграції Facebook з іншими веб-сайтами. Це допомогло сайтам стати багатими об'єктами в соціальному графі; тобто це дозволило веб-сторінкам мати ту саму функціональність, що й інші об'єкти у Facebook. Ця технологія надає можливість соціальним мережам

правильно відображати інформації про сторінку . Це стає в нагоді, коли користувач поширює посилання на сторінку в соціальній мережі, а інші користувачі можуть побачити коротку інформацію про цю сторінку не переходячи на неї, що зображено на рисунку 2.4.

Рисунок 2.4 – Результат використання технології OpenGraph

Існує чотири основні og-теги, шо є обов’язковими для заповнення: og-title, og-description, og-type, og-description.

Наступним кроком для досягнення хороших показників, що був використаний під час проектування технології є використання мікроданих[25]. Ці дані ніяк не змінюють відображення сторінки для користувача. Вони лише вказують пошуковому боту на тип інформації у певному місці гіпертекстової розмітки сторінки. Чудовим прикладом використання мікророзмітки є позначення блоку з відповідями на питання, що часто задаються. Приклад цього зображено на рисунку 2.5.

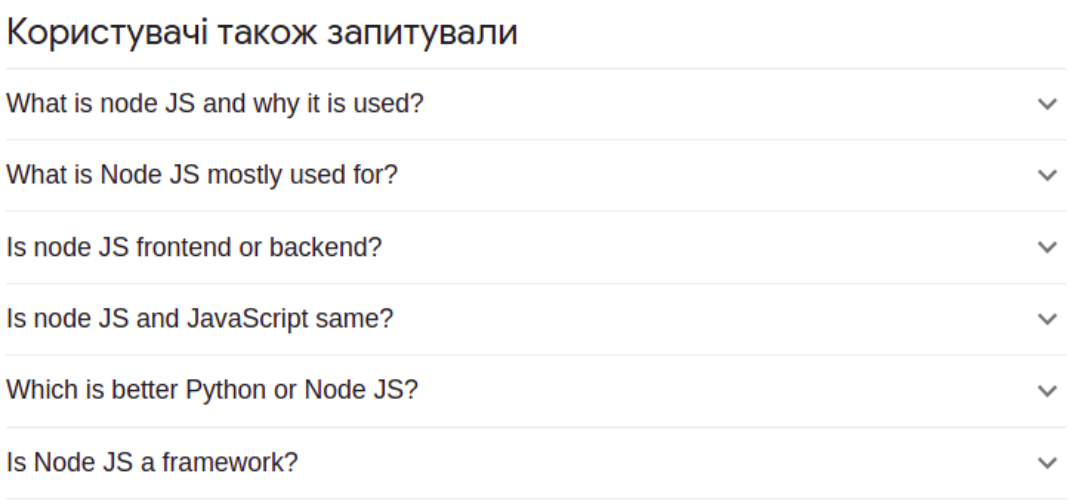


Рисунок 2.5 – Результат використання мікроданих

Ще однією технологією, що допомагає підняти рейтинги сайту є AMP[26]. Це технологія для оптимізації веб сторінок під мобільні пристрої, що була розроблена компанією google в 2015 році.

Технологія складається з таких компонентів:

– AMP HTML — мова HTML, в якому частина тегів замінена на еквівалентні AMP-теги, а частина заборонена для використання.

– AMP JS — в роботі використовується власна JS -Бібліотека, що дозволяє елементам сторінки завантажуватися асинхронно.

– Google AMP Cache — в процесі індексації AMP-сторінки, пошукова система кешує її дані і відтворює зі своїх серверів.

Технологія AMP ставить перед розробником велику кількість обмежень, але гарантує, що у випадку, коли всі умови було дотримано сторінка буде мати кращі рейтинки. Про важливість цієї технології говорить ще те, що до літа 2021 року сторінки, що були створені з використанням технології AMP у пошуку відображалися з спеціальною позначкою у вигляді блискавки, що зображено на рисунку 2.6.

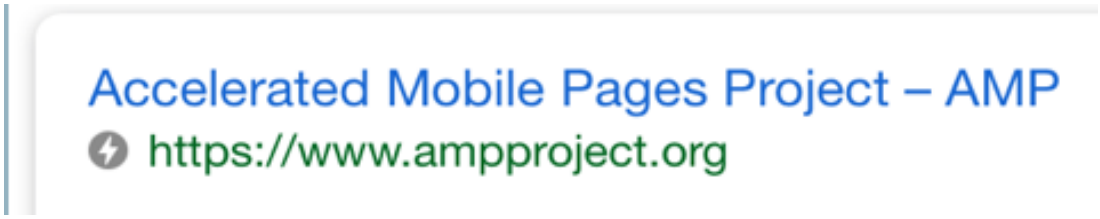


Рисунок 2.6 – Результат технології AMP

У проектованій системі моделювання пропонується вводити ці дані під час створення сутності сторінки, а також надається можливість редагування цієї інформації у будь який момент часу.

Заповнення мета-інформації та робота з контентом системи є єдиними легальними способами роботи з пошуковою оптимізацією всередині системи, проте інколи цього буває недостатньо для досягнення хороших показників. Враховуючи це в проектованій системі моделювання веб-орієнтованих систем створений спеціальний функціонал, що перевіряє кожного користувача, що входить у систему. У випадку, коли таким користувачем є пошуковий бот, сервер повертає йому спеціальну версію сторінки у якій повністю відсутній клієнтський код. Така перевірка є досить легкою у реалізації оскільки у кожному запиті до серверу присутній заголовок “User-Agent” у якому описана основна інформація про систему користувача. Заголовок, що приходить від звичайного користувача відрізняється від заголовку що приходить від боту, що дає можливість легкого розпізнання та приховування певної інформації. Це пришвидшує швидкість завантаження сторінки, що також є важливим показником під час ранжування сторінки.

Використання всіх описаних вище відомих технологій та засобів роботи з пошуковою оптимізацією у поєднанні з нелегальним підходом до рендерингу сторінок на сервері гарантує хороші показники, а отже хорошу конверсію користувачів.

Висновки до розділу 2.

В другому розділі, було удосконалено метод рендерингу сторінки на боці серверу, що пришвидшує швидкість роботи системи. Також було покращено загальну схему роботи з пошуковою оптимізацією сторінки за рахунок розробленої функції, що приховує деякий контент сторінки від пошукових ботів. Удосконалення підходів до рендерингу сторінки на сервері та пошукової оптимізації сторінки є основою для створення ефективної технології для моделювання веб-орієнтованих систем.

Розділ 3

Проектування панелі адміністратора з можливістю моделювання веб-орієнтованої системи.

3.1 Вибір технологій розробки.

Для розробки ефективної системи дуже важливим кроком є вибір технологій. Цей вибір є дуже важливим, оскільки впливає на якість роботи системи та швидкість її створення. Для розробки технології моделювання web-орієнтованої системи, спираючись на власний досвід, можливості систем та тенденції ринку, було обрано такий перелік технологій:

– ОС — будь яка(вибір операційної системи має велике значення тільки на етапі розгортання продукту);

– Сервер — nginx[27];

– База даних — postgreSQL[28]:

– Фреймворк серверної частини — adonisjs[29];

Nginx - це безкоштовний високопродуктивний HTTP-сервер із відкритим вихідним кодом, а також проксі-сервер IMAP/POP3. NGINX відомий своєю високою продуктивністю, стабільністю, багатим набором функцій, простою конфігурацією та низьким споживанням ресурсів.

NGINX - один із небагатьох серверів, написаних для вирішення проблеми C10K . На відміну від традиційних серверів, NGINX не покладається на потоки для обробки запитів. Натомість він використовує значно більш масштабовану (асинхронну) архітектуру, керовану подіями. Ця архітектура використовує невеликі, але, що більш важливо, передбачувані обсяги пам'яті під навантаженням. Навіть якщо ви не очікуєте, що будете обробляти тисячі одночасних запитів, ви все одно отримаєте вигоду від високопродуктивної та невеликої пам'яті NGINX. NGINX масштабується у всіх напрямках: від найменших VPS аж до великих кластерів серверів.

Мета NGINX полягала в тому, щоб створити найшвидший веб-сервер, і підтримувати цю досконаліть все ще є центральною метою проекту . NGINX постійно перевершує Apache та інші сервери в тестах для вимірювання продуктивності веб-сервера . Однак після вихідного випуску NGINX веб-сайти розширилися від простих HTML-сторінок до динамічного багатогранного вмісту. NGINX виріс разом з ним і тепер підтримує всі компоненти сучасної мережі, включаючи WebSocket, HTTP/2, gRPC і потокове передавання кількох відеоформатів (HDS, HLS, RTMP та інші).

Хоча NGINX став відомим як найшвидший веб-сервер, масштабована базова архітектура виявилася ідеальною для багатьох веб-завдань, крім обслуговування вмісту . Оскільки він може обробляти великий обсяг з’єднань, NGINX зазвичай використовується як зворотний проксі-сервер і балансувальник навантаження для керування вхідним трафіком і розподілу його на повільніші сервери вище по потоку – від застарілих серверів баз даних до мікросервісів.

NGINX також часто розміщується між клієнтами та другим веб-сервером, щоб служити термінатором SSL/TLS або веб-прискорювачем. Виступаючи як посередник, NGINX ефективно справляється із завданнями, які можуть сповільнити роботу вашого веб-сервера, наприклад, узгодження SSL/TLS або стиснення та кешування вмісту для підвищення продуктивності. Динамічні сайти, створені з використанням будь-чого, від Node.js до PHP, зазвичай розгортають NGINX як кеш вмісту та зворотний проксі, щоб зменшити навантаження на сервери додатків і максимально ефективно використовувати базове обладнання.

```
server {

  listen 80 default_server;
  listen [::]:80 default_server;

  server_name localhost;

  access_log off;
  error_log off;

  client_max_body_size 300m;

  charset utf-8;

  # root /vaw/www/app/public;
  # gzip on;
  # gzip_disable "msie6";
  # gzip_vary on;
  # gzip_types text/plain text/css application/json application/x-javascript text/xml application/xml application/xml+rss text/javascript application/javascript;
  # gzip_comp_level 5;

  # ssl_certificate /etc/nginx/ssl/-ssl.chained.crt;
  # ssl_certificate_key /etc/nginx/ssl/-ssl.key;

   location ~ ^/uploads/img/(?<width>\d+)x(?<height>\d+)/(?<name>.*)$ {
    proxy_pass               http://web:${NODE_PORT}/uploads/img/$name;
    # proxy_set_header         Host  $host;
    # proxy_set_header         X-Real-IP $remote_addr;
    # proxy_set_header         X-Forwarded-Proto $scheme;
    # proxy_buffer_size        512k;
    # proxy_buffers            16 512k;

    expires                  10d;
    image_filter_buffer      20M;
    image_filter             crop  $width $height;
    image_filter_webp_quality 85;
    image_filter_sharpen     100;
  }

  # location ~ ^/(api|admin|docs|uploads|download|swagger.json) {
  location / {
    proxy_pass               http://web:${NODE_PORT}/;
    proxy_http_version       1.1;
    proxy_set_header         Upgrade $http_upgrade;
    proxy_set_header         Connection 'upgrade';
    proxy_set_header         Host $host;
    proxy_set_header         X-Real-IP $remote_addr;
    proxy_set_header         X-Forwarded-Proto $scheme;
    proxy_set_header         X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_cache_bypass       $http_upgrade;
  }
}
```

Рисунок 3.1 – Конфігурація серверу nginx

PostgreSQL - це розширена реляційна база даних з відкритим кодом корпоративного класу, яка підтримує SQL (реляційні) та JSON (нереляційні) запити. SQL — декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації, системи контролю за доступом до бази даних. PostgreSQL є надзвичайно стабільною системою управління базами даних, яка підтримується більш ніж 20-річним розвитком громади, що сприяло її високому рівню стійкості, доброчесності та коректності. PostgreSQL використовується як основне сховище даних або сховище даних для багатьох веб, мобільних, геопросторових та аналітичних додатків. Остання основна версія - PostgreSQL 12.

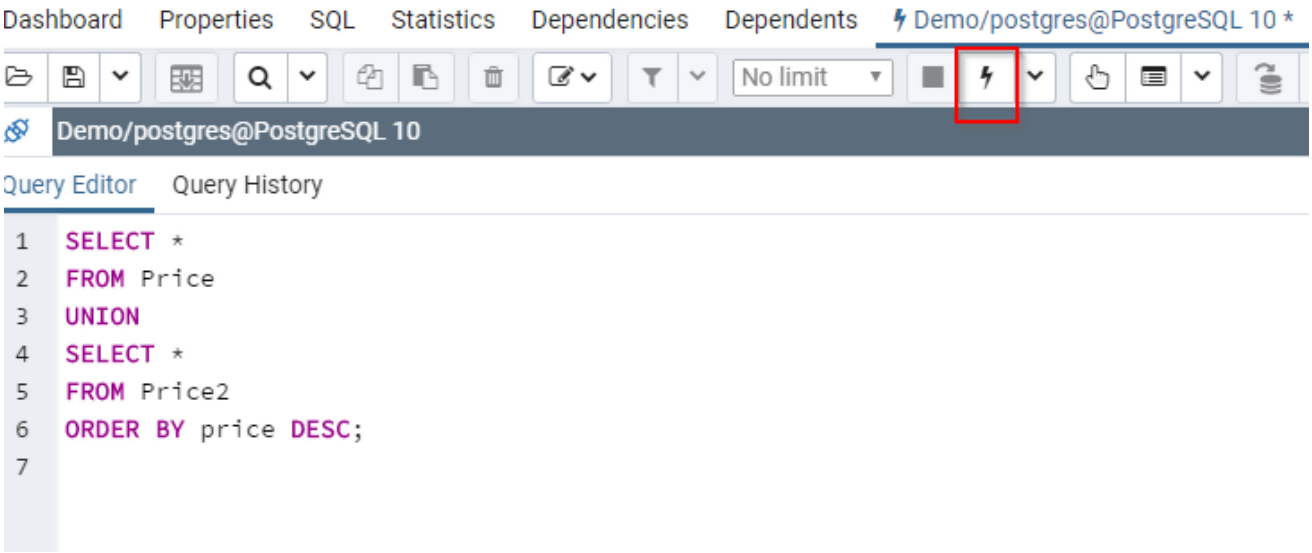


Рисунок 3.2 – Приклад запиту до бази даних

AdonisJS - це платформа Node.js, орієнтована на ергономіку, стабільність та швидкість розробників. AdonisJS написаний з нуля з чітким принципом і цілями, щоб бути міцною інтегрованою системою. Він також слідує тому ж принципу MVC, що використовується у багатьох популярних фреймворках, таких як Laravel, Rails та Spring. Він зосереджений на користувацькому досвіді розробників, стабільності та швидкості. Як повноцінна веб -платформа, AdonisJS має вбудований набір функції та модулів, які якісно відрізняють його серед інших фреймворків Node.js.

Adonisjs має вбудовану систему маршрутизації, яка дозволяє описувати як статичні шляхи до певних ресурсів, так і динамічні шляхи з одним або кількома параметрами. Ще однією з переваг adonisjs є можливість створення middleware, тобто функцій проміжного опрацювання, такі функції є досить важливою частиною майже будь якої web-орієнтованої системи. Незважаючи на те, що зараз є дуже популярним використання односторінкових застосунків(SPA), де весь контент сторінки змінюється можливостями мови JavaScript без перезавантаження, що є дуже хорошим рішенням для складних користувацьких інтерфейсів, проте рендеринг на стороні серверу досі має свої сфери застосування, а саме на сторінках ресурсу, які повинні добре індексуватися пошуковими системами. Adonisjs має вбудований шаблонізатор HTML, що дозволяє йому

виконувати рендерінг на стороні серверу. Також до переваг цього фреймворку можна віднести наявність будівельника SQL запитів.

Валідація вхідних даних є дуже важливою частиною будь якого серверного застосунку і adonisjs має вбудований валідатор даних з багатьма функціями перевірки. Крім перелічених вище adonisjs має ще декілька модулів, які значно спрощують написання коду в порівнянні з його аналогами.

```
import Route from '@ioc:Adonis/Core/Route'

Route.get('/', () => {
  return 'Hello world'
})
```

Рисунок 3.3 – Приклад коду написаного на adonisjs

Після вибору технологій розробки необхідно організувати ефективну роботу з ними. Налаштування цих засобів окремо на комп'ютері кожного розробника буде займати багато часу. Щоб вирішити цю проблему необхідно запакувати ці технології в єдиний контейнер, який потім можна буде скачати і запустити на будь якій системі. Ця проблема не є новою і в теперішній час існують інструменти які її вирішують. Найкращим інструментом для цього є Docker[30](рис 2.2).

Рисунок 3.4 – Логотип docker

Docker - це платформа контейнеризації з відкритим кодом. Він дозволяє розробникам упаковувати програми в контейнери - стандартизовані виконувані компоненти, що поєднують вихідний код програми з бібліотеками операційної системи (ОС) та залежності, необхідні для запуску цього коду в будь -якому середовищі. Контейнери спрощують доставку розподілених додатків і стають все більш популярними, коли організації переходять на хмарну розробку та гібридні багатохмарні середовища. Docker - це, по суті, набір інструментів, який дозволяє розробникам створювати, розгортати, запускати, оновлювати та зупиняти контейнери за допомогою простих команд з метою автоматизації економії часу за допомогою єдиного API.

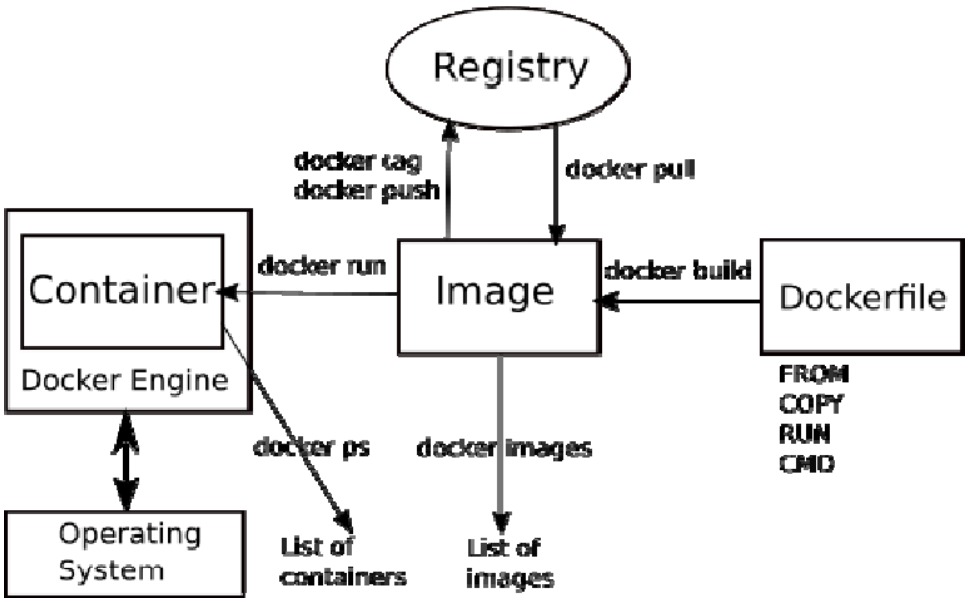


Рисунок 3.5 – Структура docker

Контейнери стали можливими завдяки можливостям ізоляції процесів та віртуалізації, вбудованим у ядро Linux. Ці можливості, такі як групи керування (групи C) для розподілу ресурсів між процесами та простори імен для обмеження доступу або видимості процесів до інших ресурсів або областей системи, - дозволяють декільком компонентам програми спільно використовувати ресурси одного екземпляра хоста, що працює система приблизно так само, як гіпервізор дозволяє кільком віртуальним машинам (віртуальним

машинам) ділитися процесором, пам'яттю та іншими ресурсами одного апаратного сервера.

В результаті технологія контейнерів пропонує всі функціональні можливості та переваги віртуальних машин, включаючи ізоляцію додатків, економічно масштабоване масштабування та одноразове використання, а також важливі додаткові переваги:

– Менша вага: На відміну від віртуальних машин, контейнери не несуть корисного навантаження всього екземпляра ОС та гіпервізора; вони включають лише процеси та залежності ОС, необхідні для виконання коду. Розміри контейнерів вимірюються в мегабайтах (порівняно з гігабайтами для деяких віртуальних машин), покращують використання місткості обладнання та мають швидший час запуску.

– Більша ефективність використання ресурсів: За допомогою контейнерів ви можете запустити в кілька разів більше копій програми на одному обладнанні, ніж за допомогою віртуальних машин. Це може зменшити ваші хмарні витрати.

– Покращена продуктивність розробників: Порівняно з віртуальними машинами, контейнери швидше і простіше розгортати, надавати та перезапускати. Це робить їх ідеальними для використання в системах безперервної інтеграції та безперервної доставки (CI/CD), а також краще підходить для команд розробників, які впроваджують практики Agile та DevOps .

Для контейнеризації усіх необхідних сервісів необхідно описати свій Dockerfile(файл з інструкціями для запуску) для кожного з них та створити один docker-compose файл у якому будуть описуватися загальні правила запуску програмного засобу. Compose-це інструмент для визначення та запуску багато контейнерних програм Docker. За допомогою Compose ви використовуєте файл YAML для налаштування послуг вашої програми.

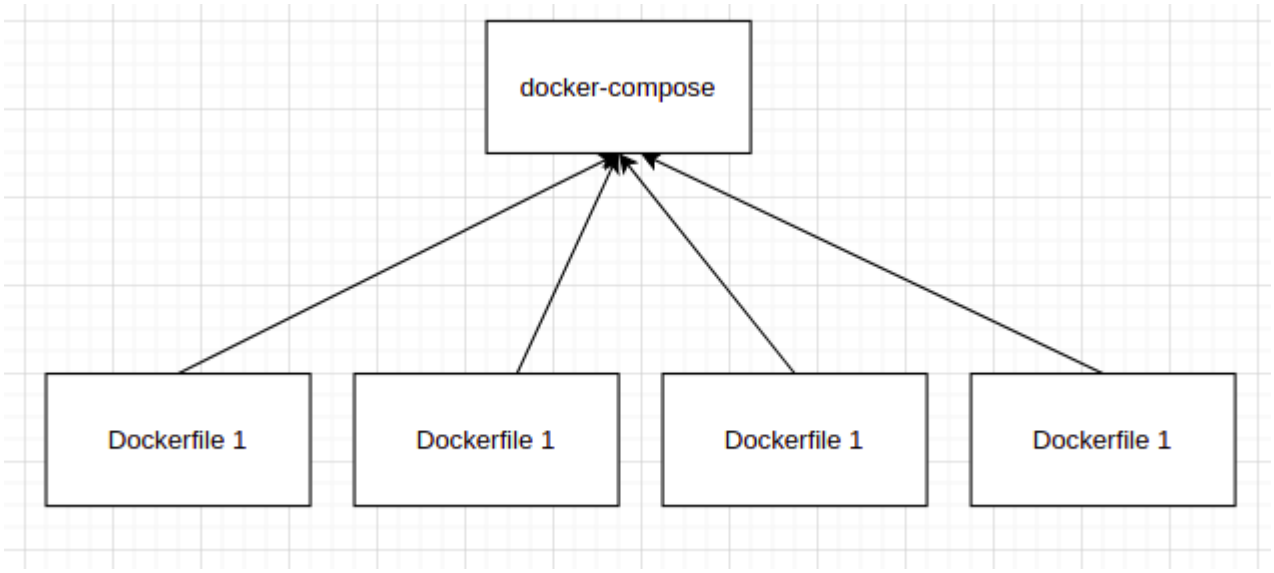


Рисунок 3.6 – Схема роботи docker-compose

```
FROM nginx:alpine

RUN apk update && apk add --update --no-cache \
  nginx-mod-http-image-filter \
  libgd \
  sudo \
  bash

#  nginx-mod-http-image-filter
RUN echo -e "load_module /etc/nginx/modules/ngx_http_image_filter_module.so;\n$(cat /etc/nginx/nginx.conf)" > /etc/nginx/nginx.co

ADD ./docker/nginx/default.conf /etc/nginx/conf.d/default.conf
```

Рисунок 3.7 – Dockerfile для nginx

```
FROM postgres:12.5

# If you need POSTGRES_MULTIPLE_DATABASES
# COPY ./docker/postgres/create-multiple-postgresql-databases.sh /docker-entrypoint-initdb.d

# If you need install DB extensions
COPY ./docker/postgres/extensions.sql /docker-entrypoint-initdb.d
```

Рисунок 3.8 – Dockerfile для postgreSQL

```
FROM node:12

WORKDIR "/home/node/app"

COPY package*.json ./

RUN npm i -g @adonisjs/cli pm2
```

Рисунок 3.9 – Dockerfile для adonisjs

```
version: "3.3"

services:
  db:
    build:
      context: ./
      dockerfile: docker/postgres/Dockerfile
    environment:
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_USER: ${DB_USER}
      POSTGRES_DB: ${DB_DATABASE}
    ports:
      - ${EXTERNAL_DB_PORT}:${DB_PORT}
  web:
    user: 1000:1000
    build:
      context: ./
      dockerfile: docker/web/Dockerfile
    volumes:
      - ./:/home/node/app
    command: bash -c 'while !</dev/tcp/db/${DB_PORT}; do sleep 1; done; npm run adonis-serve-dev'
  nginx:
    build:
      context: ./
      dockerfile: docker/nginx/Dockerfile
    volumes:
      - ./docker/nginx/ssl:/etc/nginx/ssl:ro
      - ./docker/nginx/default.conf:/etc/nginx/conf.d/default.template:ro
    depends_on:
      - db
      - web
    ports:
      - ${EXTERNAL_NGINX_PORT_HTTP}:80
      - ${EXTERNAL_NGINX_PORT_HTTPS}:443
    environment:
      NGINX_EXTERNAL_HOST_HTTP: ${EXTERNAL_NGINX_PORT_HTTP}
      NGINX_EXTERNAL_HOST_HTTPS: ${EXTERNAL_NGINX_PORT_HTTPS}
      NODE_PORT: ${PORT}
    command: bash -c "envsubst '$${NGINX_EXTERNAL_HOST_HTTP},$${NGINX_EXTERNAL_HOST_HTTPS},$${NODE_PORT}' < /etc/nginx/conf.d/default.template > /etc/nginx/conf.d/default.conf && exec nginx -g 'daemon off
```

Рисунок 3.10 – docker-compose.yml

Після написання даних інструкцій запуск програмного продукту можна виконувати тільки однією командою, яка автоматично запустить всі необхідні сервіси всередині ізольованого контейнера. Також docker дозволяє не перейматися про версії того чи іншого сервісу. Docker скачує версію програмного засобу яка описана в Dockerfile, що гарантує запуск системи на будь якому комп’ютері і дозволяє розробникам економити велику кількість часу.

```
docker-compose up
Starting rexsoftinc_db_1  ... done
Starting rexsoftinc_web_1 ... done
Starting rexsoftinc_nginx_1 ... done
Attaching to rexsoftinc_db_1, rexsoftinc_web_1, rexsoftinc_nginx_1
```

Рисунок 3.11 – Приклад запуску контейнеризованої системи

Після виконання описаних вище дій маємо проміжний результат — базовий шаблон web-орієнтованої системи.

3.2 Проектування інформаційної технології моделювання web-орієнтованої системи.

Основою кожної системи є база даних. Для створення динамічної сторінки, що формується з панелі адміністратора було розроблено схему бази даних, яка складається із двох основних таблиць: “pages” та “components”. Схему цієї частини бази даних зображено на рисунку 3.12. У таблиці “pages” зберігається назва сторінки, унікальний ідентифікатор для побудови посилання, мета-інформація та булеве значення, що відповідає за те чи сторінка доступна чи ні.

У таблиці зберігається інформація про сторінку до якої відноситься компонент, тип компоненту, порядок компонента на сторінці, та json-поле “data” в якому зберігаються динамічні дані. Для створення таблиць були створені міграції бази даних(рис 3.13).

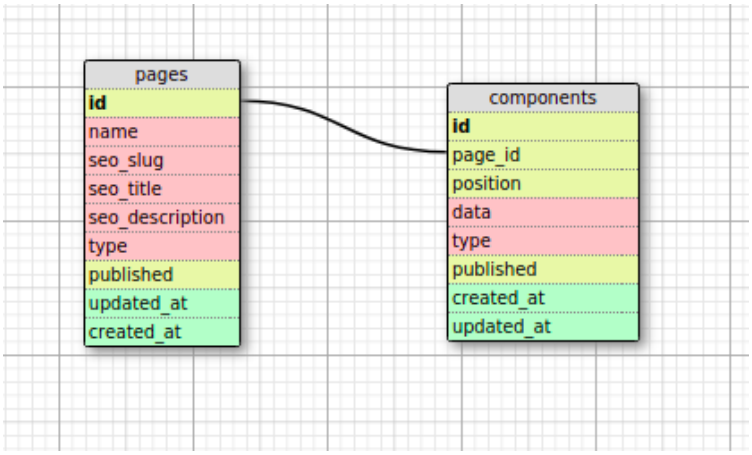


Рисунок 3.12 – Схема бази даних веб-сторінок

```
class ComponentsSchema extends Schema {
  up () {
    this.create('components', (table) => {
      table.increments()
      table.string('name')
      table.string('type')
      table.integer('position')
      table.integer('page_id').references('id').inTable('pages').onDelete('CASCADE').onUpdate('CASCADE')
      table.json('data')
      table.boolean('published').defaultTo(false);
      table.timestamps()
    })
  }
```

Рисунок 3.13 – Міграції бази даних для таблиці компонентів

Окрім таблиць, що необхідні для побудови сторінки, було створено таблицю в якій зберігається інформація про навігацію системи, що моделюється. Ця таблиця називається “navigation” її схему зображено на рисунку 3.14.

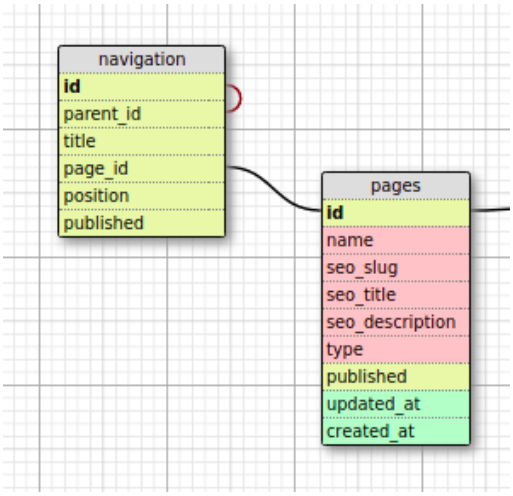


Рисунок 3.14 – Схема таблиці “navigation” та її зв’язок із таблицею “pages”

Дуже цікавим елементом цієї таблиці є зовнішній ключ, який посилається на запис із цієї ж таблиці. Таке рішення дає можливість створювати вкладеність, що є дуже важливо для навігації у веб-системі.

Крім таблиць, що були описані вище, було створено таблиці для зберігання адміністраторів, ролей та прав.

Для зручної роботи із базою даних було описано моделі(Рисунок 3.15), тобто було використано шаблон проектування під назвою “ActiveRecord”.

Шаблон active record — це шаблон проектування що використовується при реалізації доступу до реляційних баз даних. У моделях бази даних описуються взаємозв’язки, методи життєвого циклу.

```
class Page extends Model {
  static boot () {
    super.boot()
    this.addHook('afterSave', async (instance) => {
      await instance.load('allComponents')
      await Promise.all(instance.getRelated('allComponents').rows.map(async (component) => {
        if(Config.get(`admin.components.${component.type}`).not_allowed.includes(instance.type)) {
          component.published = false;
          return await component.save();
        }
        return component;
      }))
    })
  }

  static get table() {
    return 'pages';
  }

  allComponents() {
    return this.hasMany('Pages/Models/Component', 'id', 'page_id')
  }

  components() {
    return this.hasMany('Pages/Models/Component', 'id', 'page_id').where('published', true).orderBy('position', 'asc')
  }
```

Рисунок 3.15 – Модель для роботи із таблицею “pages”

Схему взаємодії кожної сутності з бази даних при побудові динамічної сторінки зображено на рисунку 3.16.

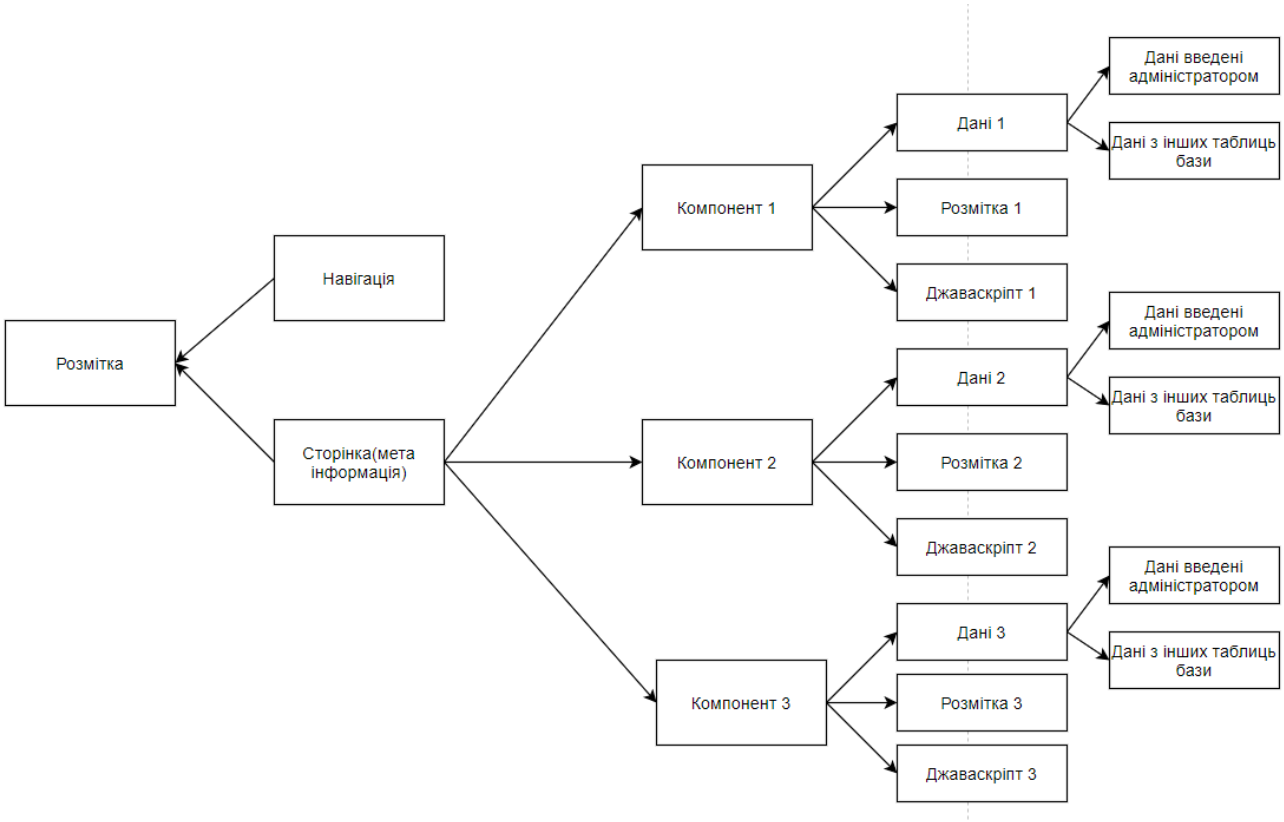


Рисунок 3.16 – Модель взаємодії сутностей бази даних при побудові динамічної сторінки

3.3 Розробка інформаційної технології моделювання web-орієнтованої системи

Після створення схеми бази даних було прийнято рішення розробити власну архітектуру на основі тієї, що вже була реалізована у фреймворку Adonisjs. Архітектура — це принцип організації коду системи. Для зручної роботи з кодом було вирішено розділити його на модулі. В кожному модулі будуть зберігатися власні контроллери, маршрутизатори, проміжні обробники, конфігураційні файли, розмітка та інше. Для зручності роботи із такою архітектурою було написано код, який автоматично реєструє модулі у сервері node.js. Цей код складається функцій для реєстрації розмітки, конфігурації, маршрутизації, побудови меню адмін панелі.

```
registerConfigs(modulePath, moduleName) {
  const configsPath = path.resolve(modulePath, moduleName, 'Configs');
  if (fs.existsSync(configsPath)) {
    const moduleMenu = requireAll(configsPath);
    const { _config } = Config;

    Object.assign(Config, { _config: deepmerge(_config, moduleMenu) });
  }
}
```

Рисунок 3.17 – Код для роботи із конфігурацією

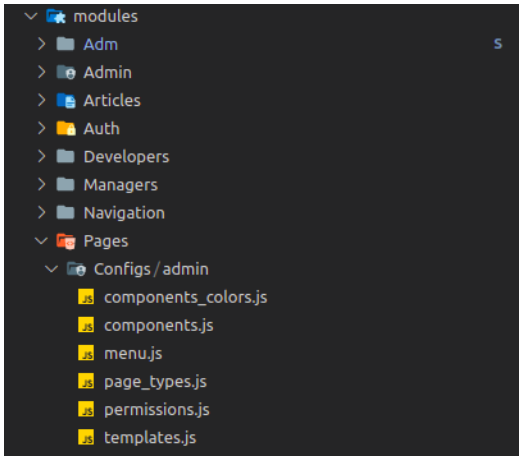


Рисунок 3.18 – Розроблена архітектура організації коду

Наступним кроком розробки є наповнення бази даних початковими сутностями. Для цього прописується спеціальний код, який виконується при першому запуску програми. До початкових даних відносяться: базові ролі панелі адміністратора, доступи та перший адміністратор. Приклад коду для заповнення бази даних:

```
const Env = use('Env');
const AdminUser = use('Adm/Models/AdminUser');
class AdminUsersSeeder {
     async run() {
                 await AdminUser.create({
                 password: Env.get('ADMIN_PASSWORD', '123123'),
                 email: Env.get('ADMIN_EMAIL', 'admin@admin.com'),
           });
     }
```

```
}
module.exports = AdminUsersSeeder;
```

Для надання доступів адміністратору було написано спеціальну функцію. Ця функція збирає інформацію про доступи, яка зберігається у модулях, додає їх до бази та приєднує їх до адміністратора. Приклад використання функції зображено на рисунку 3.17. Код функції:

```
async handle() {
      const existedPermissions = await AdminPermission.pair('slug', 'name');
      const allConfigPermissions = [].concat.apply([],
Object.entries(Config.get('admin.permissions')).map((g) => g[1]));
      for(const permission of allConfigPermissions) {
            const { slug, name } = permission;
            this.info(slug);
            if(Object.prototype.hasOwnProperty.call(existedPermissions, slug))
      {
            await AdminPermission.query().where('slug', slug).update({ slug,
      name });
            } else {
                  await AdminPermission.create({ slug, name });
            }
      }
      const onDeleteCount = await AdminPermission.query().whereNotIn('slug',
allConfigPermissions.map((p) => p.slug)).delete();
      if (onDeleteCount) this.success(`onDelete (${onDeleteCount}) -> done`);
      const existedPermissionsReload = await AdminPermission.pair('id',
'slug');
      const existedPermissionsReloadIds =
Object.keys(existedPermissionsReload);
      const adminUser = await AdminUser.findBy({ email:
Env.get('ADMIN_EMAIL', 'admin@admin.com') });
      await adminUser.permissions().detach();
      await adminUser.permissions().attach(existedPermissionsReloadIds);
      Database.close();
      return true;
}
```

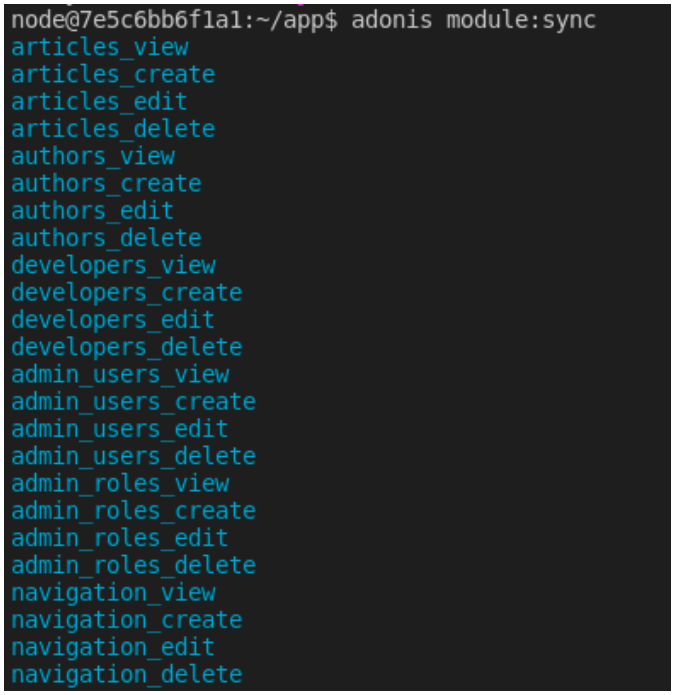


Рисунок 3.19. Використання команди для надання доступів адміністратору

Наступним етапом розробки системи є власне створення модулів роботи із навігацією та сторінками. У цих модулях повинні бути реалізовані функції для додавання, перегляду, редагування та видалення записів. Для відображення даних, що знаходяться у таблиці було використано бібліотеку під назвою Datatables. Використання цієї бібліотеки вимагає написання сервісу для опрацювання запитів. У цьому сервісі повинні бути реалізовані функції для пагінації, пошуку, сортування даних. Для створення та редагування даних було прийнято рішення використовувати модальні вікна.

Модуль роботи з динамічними веб-сторінками є досить складним та складається із багатьох функцій та налаштувань. Для того щоб не навантажувати інтерфейс було прийнято рішення розділити процес створення сторінки на декілька кроків:

– створення сторінки та заповнення мета-інформації;
– перехід до списку компонентів сторінки;
– додавання нового елементу на сторінку;

– редагування даних елементу;

Такий процес створення та наповнення сторінки є дуже простим у використанні та не потребує ніяких додаткових навичок. Для того, щоб реалізувати можливість кожному компоненту мати власний файл стилів та власний файл із клієнтським кодом, було прийнято рішення створювати окрему папку для кожного із таких компонентів, що зображено на рисунку 3.19. У цій папці обов'язково знаходиться розмітка, яка повертається користувачу. Окрім цієї розмітки у папці ще можуть знаходитись: форма для заповнення інформації, файл із стилями, файл із клієнтським кодом.

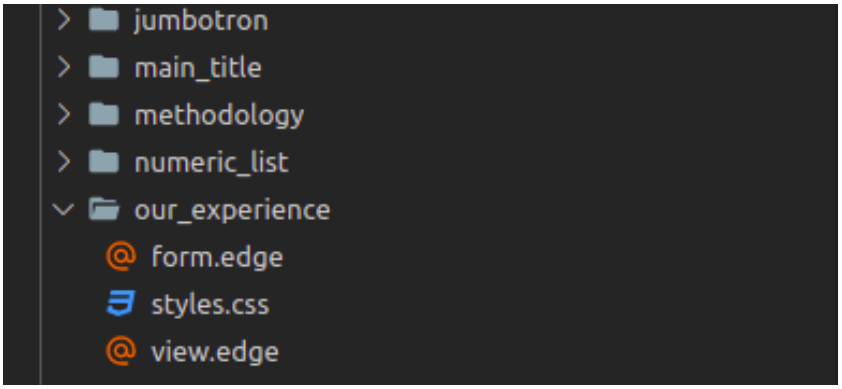


Рисунок 3.20 – Структура папки компонентів

Кожний компонент має свої налаштування, які зберігаються у відповідному конфігураційному файлі. У цьому конфігураційному файлі вказується на яких типах сторінок цей компонент може використовуватися, тип цього компоненту(статичний чи динамічний), чи потрібно завантажувати стилі, запити до бази даних, які необхідно зробити або під час рендерингу, або під час редагування інформації. Приклад конфігурації одного із компонентів:

```
developers: {
        title: 'Developers',
        type: 'dynamic',
        image: '/assets/previews/support_packages.png',
        styles: true,
        scripts: [
```

```
        `<script async custom-element="amp-base-carousel"
src="https://cdn.ampproject.org/v0/amp-base-carousel-0.1.js"></script>`
        ],
        not_allowed: [],
        queries: {
                form: {
                        categories: async () => {
                                return Category.query()
                                        .withCount('developers as developers')
                                        .fetch();
                        }
                },
                view: {
                                developers: async (page, data) => {
                                if(data.category_id) {
                                        const response = await Category.query()
                                                .where('id', data.category_id)
                                                .with('developers', (builder) => {
                                                        builder.where('is_avaliable', true)
                                                        .orderBy('is_primary', 'desc')
                                                        .with('primaryCategories')
                                                        .with('secondaryCategories')
                                                })
                                                .first()
                                        return response.toJSON().developers;
                                }
                                const developers = await Developers.query()
                                        .where('is_avaliable', true)
                                .with('primaryCategories')
                                .with('secondaryCategories')
                                .fetch()
                                return developers.toJSON()
                        }
                }
        }
},
```

Наступним кроком розробки системи керування контентом є валідація вхідних даних для кожного компоненту. Для цього був створений ще один конфігураційний файл, в якому описуються правила для кожного із блоків.

```
general_top: {
        'data.title': 'required',
        'data.description': 'required',
        'data.subtitle': 'required',
        'data.background_image': 'required',
        'data.background_text': 'required',
        'data.button_link': 'required',
        'data.download_file': 'required',
        'data.download_icon': 'required',
        'data.download_text': 'required'
},
```

Ці валідаційні правила автоматично підвантажуються перед у головний валідатор перед збереженням інформації. Коли адміністратор зберігає інформацію про компонент спочатку дані потрапляють у головний валідатор, де перевіряється наявність такого компоненту в системі та підвантажуються валідаційні правила спеціально для цього компонента. Код головного валідатора:

```
const Notify = use('ADM/Notify');
const Config = use('Config');
class Save {
get rules () {
        const rules = Config.get(`
                admin.validationRules.${this.ctx.request.input('type')}
                `)
        const types = Object.keys(Config.get('admin.components')).join(',');
        return {
                id: 'required|exists:components,id',
                type: `required|in:${types}`,
                ...rules
        };
}
get sanitizationRules () {
        return {
        }
}
get validateAll () {
```

```
        return false
}
        async fails (errorMessages) {
        return this.ctx.response.json(Notify.error(errorMessages[0].message, {}))
        }
}
module.exports = Save
```

Після того, як було реалізовано можливість наповнювати систему компонентами та перевіряти вхідні дані, можна перейти до створення контроллеру, який власне буде ці дані зберігати. У контролері було описано декілька основних методів, серед яких можна виділити метод, для зміни порядку відображення компонентів на сторінці(reorder). Ще одним важливим методом є метод для редагування даних(editData), оскільки в ньому виконуються запити, які описані в конфігураційному файлі з ключем “form”. Код контроллеру:

```
class ComponentsController {
async list({ request, response, params }) {
        const { page_id } = params;
        const query = Database
        .select(
                'components.id',
                'components.name',
                'components.type',
                'components.published',
                'components.page_id',
                'components.position'
        )
        .from('components')
        .where('components.page_id', page_id)
        .orderBy('components.position', 'asc')
        const table = new Datatables(query, request);
        const res = await table.make();
        Object.assign(res, { components: Config.get('admin.components') })
        return response.json(res);
}

async edit({ response, params, __, auth }) {
```

```
        const { id, page_id } = params;
        let data = await Component.find(id);
        let page = await Page.find(page_id);
        if (id && !data) {
                return response.json(Notify.error('Not found', {}));
        }
        return response.json({
        modal: {
                title: id ?
                        __('Pages.components.edit') : __('Pages.components.create'),
                content: View.render('Pages.selectComponent', {data, page_id, page}),
                cancel: __('Adm.admin.cancel'),
                submit: __('Adm.admin.save'),
                width: 1200
        },
        success: true,
        });
}
async editData({ response, params, __, auth }) {
        const { id } = params;
        let component = await Component.find(id);
        const queries = Config.get(`admin.components.$        {component.type}.queries.form`)
        const data = {}
        if(queries) {
                await Promise.all(Object.entries(queries).map(async ([key, value]) =>                {
                        data[key] = await value(component)
                        if(data[key] && data[key].rows || data[key] instanceof Model) {
                        data[key] = data[key].toJSON()
                }
        }))
}
        if (id && !component) {
        return response.json(Notify.error('Not found', {}));
        }

        return response.json({
                modal: {
                title: id ?
                        __('Pages.components.edit') : __('Pages.components.create'),
                content: View.render(`
```

```
                    Pages.components.${component.type}.form`,
            { component, data }
                ),
                    cancel: __('Adm.admin.cancel'),
                    submit: __('Adm.admin.save'),
                    width: 1200
                },
                success: true,
            });
    }
    async save({ request, response }) {
            const input = request.only(['name', 'type', 'page_id', 'data']);
            input.published = request.input('published');
            const id = request.input('id');
            let component = {};
            if (!id) {
                const last = await Component.query().where('page_id',
input.page_id).orderBy('position', 'desc').last();
                component = await Component.create({...input, position: last ?
last.position + 1 : 1});
            } else {
                component = await Component.find(id);
                if (!component) {
                    return response.json(Notify.error('Navigation not found', {}));
                }
                component.merge(input);
                if(!await component.save()) {
                    return response.json(Notify.error('Not updated', {}));
                }
            }

            return response.json(Notify.success('Saved', {}));
    }

    async delete({ response, params }) {
            const { id } = params;
            const component = await Component.find(id);

            if (!component) {
            return response.json(Notify.error('Something went wrong. component not          found', {}));
            }
```

```
        if (await component.delete()) {
                return response.json(Notify.success('Deleted', {}));
        }
        return true;
}

async reorder({request, params, response}) {
        const { page_id } = params;
        const data = request.input('data')

        let components = await Component
                .query()
                .where('page_id', page_id)
                .whereIn('position', data.map(el => +el.old))
                .fetch();
        components = components.rows;

        await Promise.all(components.map((component) => {
                const position = data.find(el => el.old == component.position).new;
                component.position = position;
                return component.save();
        }))

        return response.json(Notify.success('Saved', {}));
}

}

module.exports = ComponentsController;
```

Після створення функціоналу для створення сторінок та наповнення їх контентом залишилось тільки реалізувати рендеринг цієї сторінки. Для реалізації цього було використано технологію розроблену у підрозділі 2.1. У цій функції буде формуватися весь необхідний для веб-сторінки код: гіпертекстова розмітка розмітка, стилі, клієнтський код, запити до бази даних. Код методу:

```
static async render(page, additionalData) {
```

```
await page.load('components');
        let pageJSON = page.toJSON();
        let styles = [];
        let scripts = [];
        if(pageJSON.components.length) {
                styles = pageJSON.components.reduce((acc, {type: cur}) => {
                        if(Config.get(`admin.components.${cur}.styles`)) {
                                return acc.indexOf(cur) === -1 ? acc.concat([cur]) : acc
                        } else {
                                return acc;
                        }
                }, [])
        scripts = pageJSON.components.reduce((acc, {type: cur}) => {
                if(Config.get(`admin.components.${cur}.scripts`)) {
                        return acc.indexOf(cur) === -1 ? acc
                        .concat(Config.get(`admin.components.${cur}.scripts`)) : acc
                } else {
                        return acc;
                }
        }, [])
        styles = await Promise.all(styles.map(async (el) => readFile(`$
{Helpers.appRoot()}/modules/Pages/resources/views/components/${el}/          styles.css`)));
        pageJSON.components = await Promise.all(pageJSON.components.map(async   ({ data, type }) => {
                data = data || {};
                const queries = Config.get(`admin.components.${type}.queries.view`)
                if(queries) {
                await Promise.all(Object.entries(queries).map(async ([key, value]) =>              {
                        data[key] = await value(page, data)
                        if(data[key] && data[key].rows || data[key] instanceof Model) {
                                data[key] = data[key].toJSON()
                        }
                }))
                }
                return View.render(`Pages.components.${type}.view`, {
                                data,
                                page: pageJSON,
                                ...additionalData
                        })
                }))
        }
        styles = [
```

```
        ... await Promise.all(['general', 'grid'].map(async (el) => readFile(`${Helpers.appRoot()}/public/css/${el}.css`))),
        ...styles
        ]
        return { page: pageJSON, styles, scripts };
  }
```

В результаті описаних вище кроків було створено повноцінну систему моделювання веб-орієнтованих систем, яка може конкурувати із технологіями-аналогами. Завдяки власному підходу до рендерингу сторінок на сервері ця технологія гарантує швидкість роботи системи навіть при великій кількості компонентів та запитів до бази даних.

Висновки до розділу 3

У третьому розділі було визначено оптимальні технології для розробки інформаційної технології для моделювання веб-орієнтованих систем. Після вибору технологій було спроектовано схему бази даних, в основі якої лежить три таблиці: navigation, pages, components. Далі було покроково описано процес створення функціоналу, що надає можливість створювати сторінки на модельованій системі та керувати їх контентом.

Розділ 4
Аналіз отриманих результатів.

4.1 Опис роботи системи.

Для входу у систему моделювання необхідно скористатися веб-браузером, в якому слід ввести домен на якому хоститься система та шлях “/admin”. В результаті виконання цих дій користувач потрапить на сторінку авторизації в системі, що зображена на рисунку 4.1. На цій сторінці розміщена форма, що складається із двох полів для заповнення. Цими полями є адреса поштової скриньки користувача та його пароль. Кожне із цих полів є обов'язковим для заповнення.

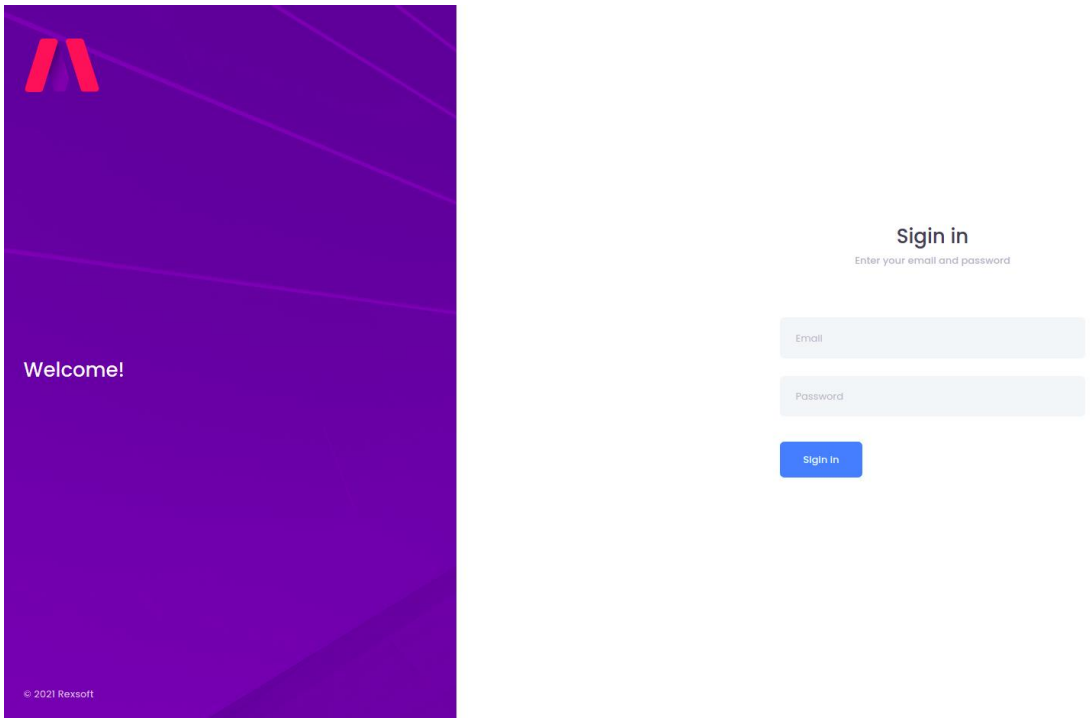


Рисунок 4.1 – Екран входу в систему

Після введення даних необхідних для авторизації, користувач потрапляє до панелі адміністратора системи. У меню панелі адміністратора користувач бачить розділи відповідно до його прав у системі, що відображено на рисунках 4.2, 4.3.

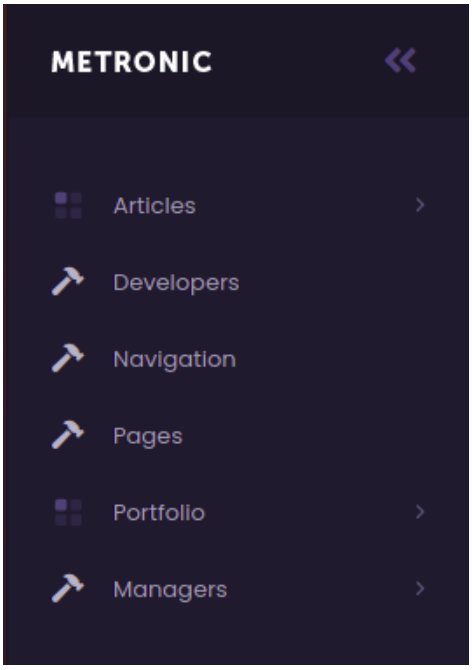


Рисунок 4.2 – Меню із розділами, що доступні адміністратору системи

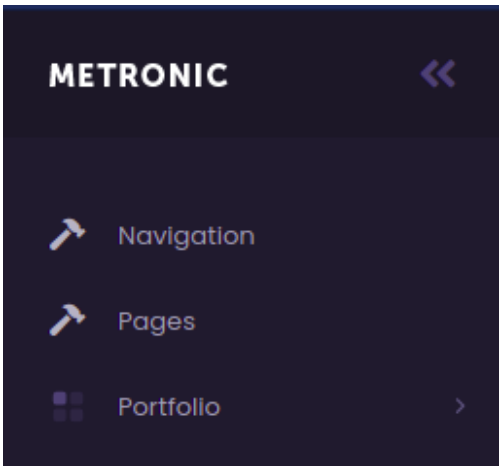


Рисунок 4.3 – Меню із розділами, що доступні модератору системи

Дозволи користувачів формуються у відповідності до ролі. Ролі можна налаштувати та редагувати всередині системи, що продемонстровано на рисунку 4.4

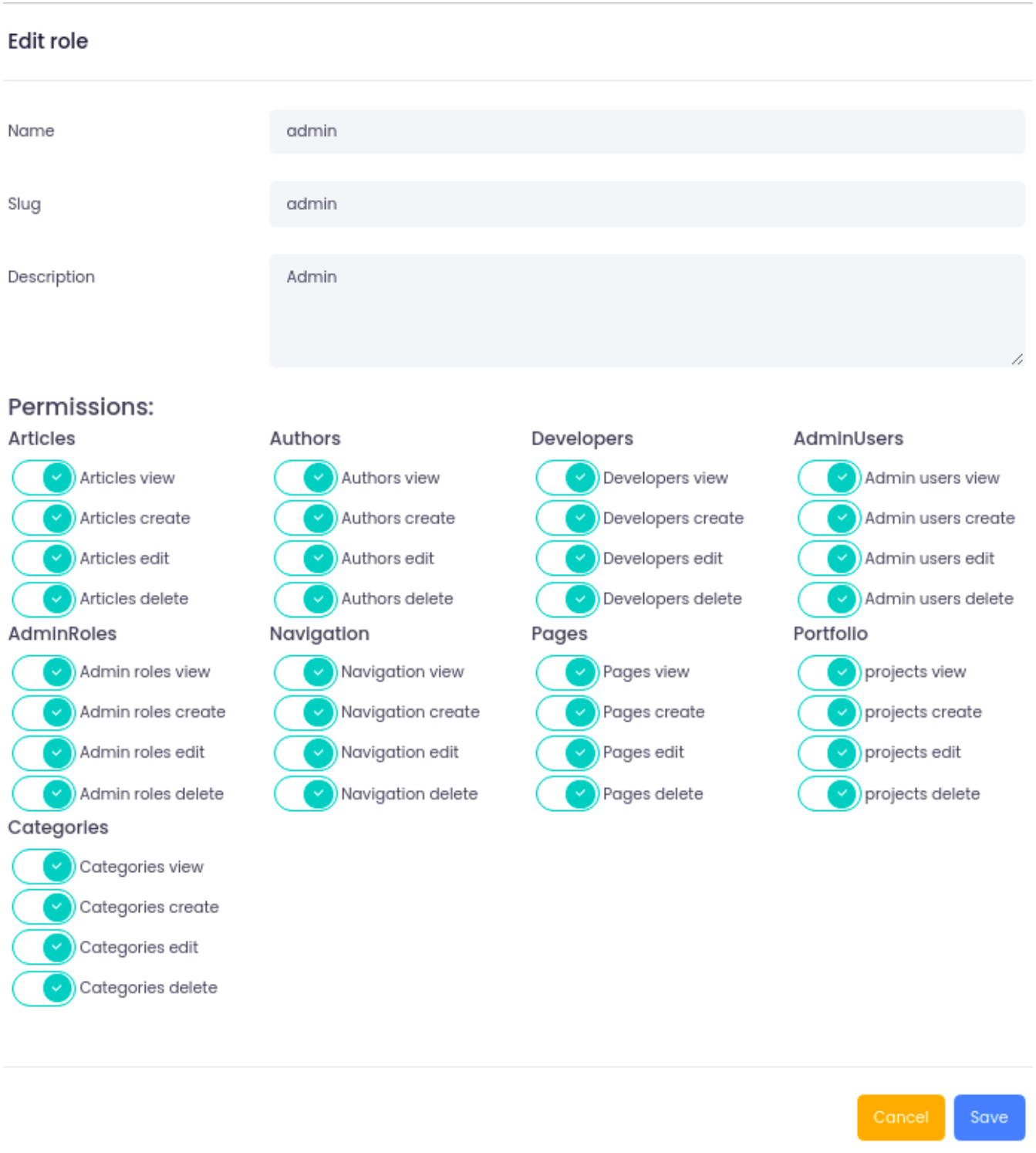


Рисунок 4.4 – Форма редагування ролі в системі

Для створення нової сторінки в системі потрібно перейти до розділу “Pages”. При переході на сторінку цього розділу користувач бачить таблицю із списком сторінок, що були створені в системі, кнопку для створення нової сторінки, та кнопки, що відкривають контекстне меню для кожної конкретної сторінки(Рисунки 4.5,4.6).

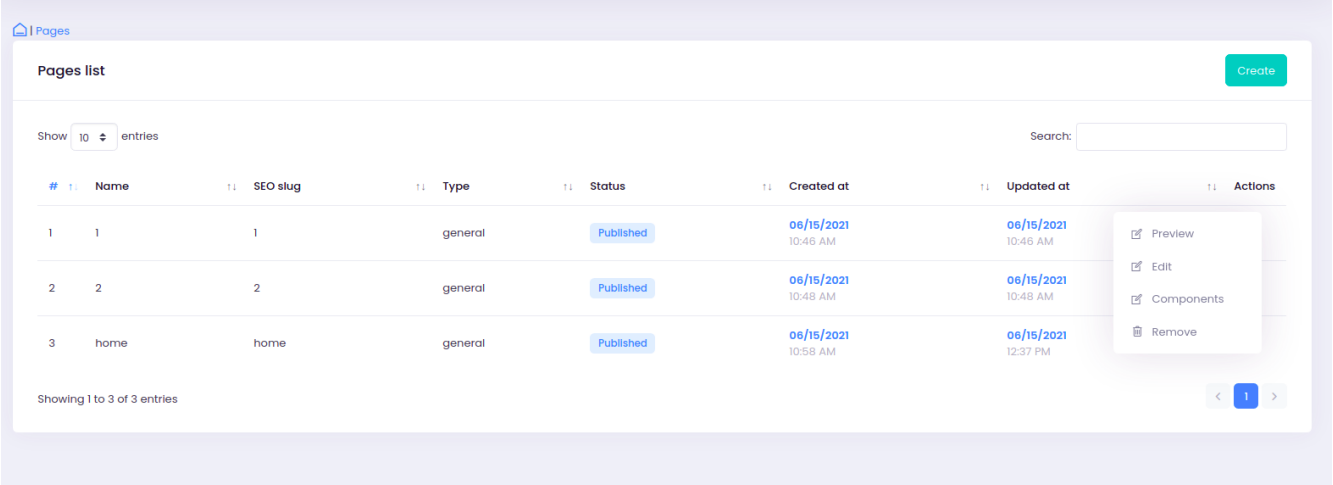


Рисунок 4.5 – Вигляд розділу “Pages”

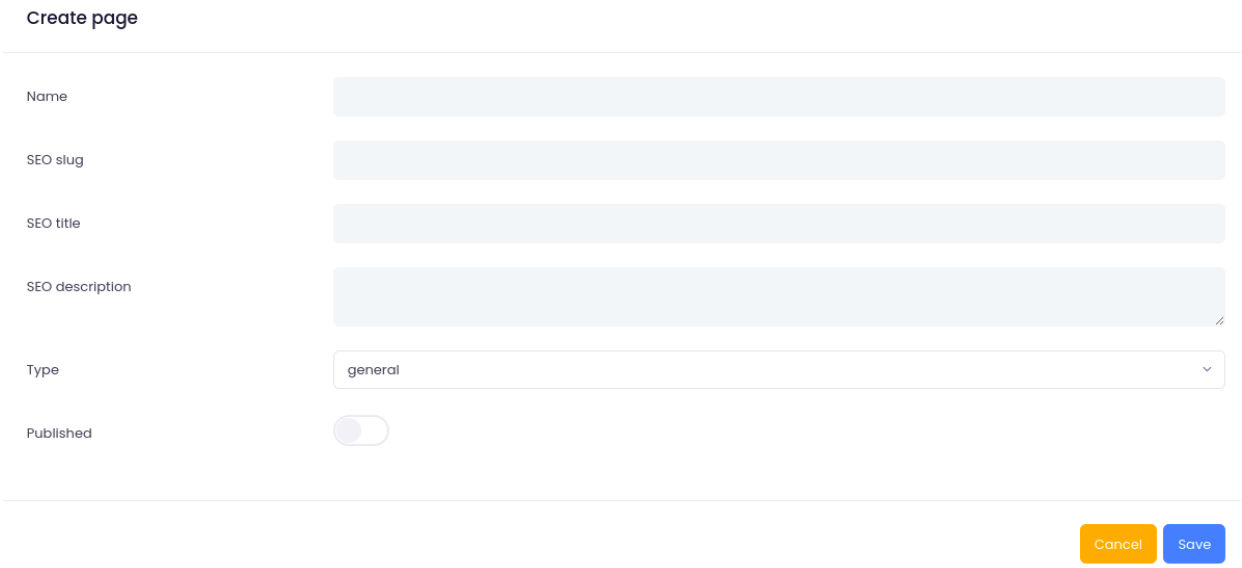


Рисунок 4.6 – Форма створення нової веб-сторінки

Після заповнення форми зображеної на рисунку 4.6 відбувається збереження сторінки у базі даних та автоматичне оновлення списку веб-сторінок на екрані. Наступним кроком є заповнення сторінки контентом, для цього треба викликати меню сторінки та перейти на пункт “components”. Натиснувши на цю кнопку користувач потрапляє на сторінку із списком компонентів, що використовуються на сторінці. Також на цій сторінці є кнопки для додавання нових компонентів, попереднього перегляду сторінки та редагування інформації самої сторінки. Вигляд розділу “components” зображено на рисунку 4.7.

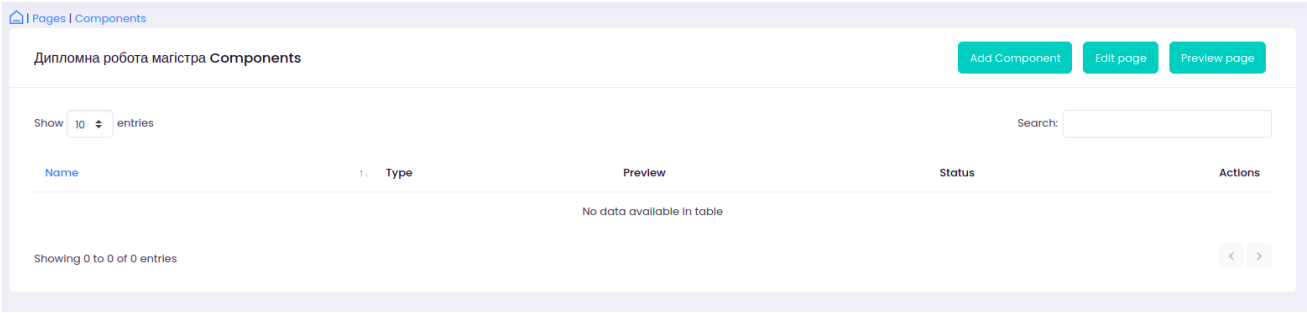


Рисунок 4.7 – Список компонентів сторінки

На даному етапі список компонентів сторінки порожній. Для додання компоненту потрібно натиснути кнопку “add component” і, у вікні що з’явилося, вибрати тип компонента, ввести назву його назву. Цей функціонал зображено на рисунку 4.8.

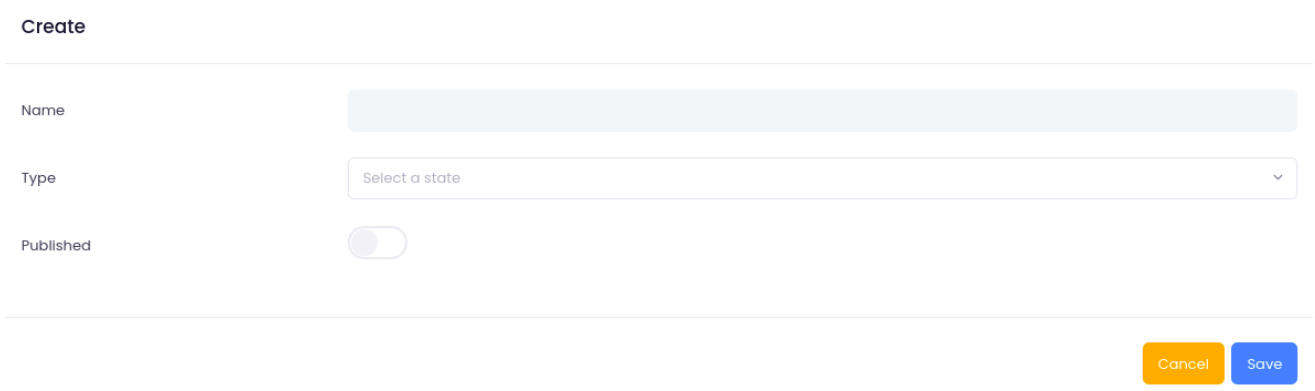


Рисунок 4.8 – Додавання нового компонента

Після додавання компоненту на сторінку потрібно наповнити його контентом натиснувши кнопку “edit data”. Після натискання на цю кнопку відкриється форма (Рисунок 4.9), що відповідає обраному типу компонента(кожен компонент має власну форму для заповнення даними оскільки потребує конеретного набору інформації).

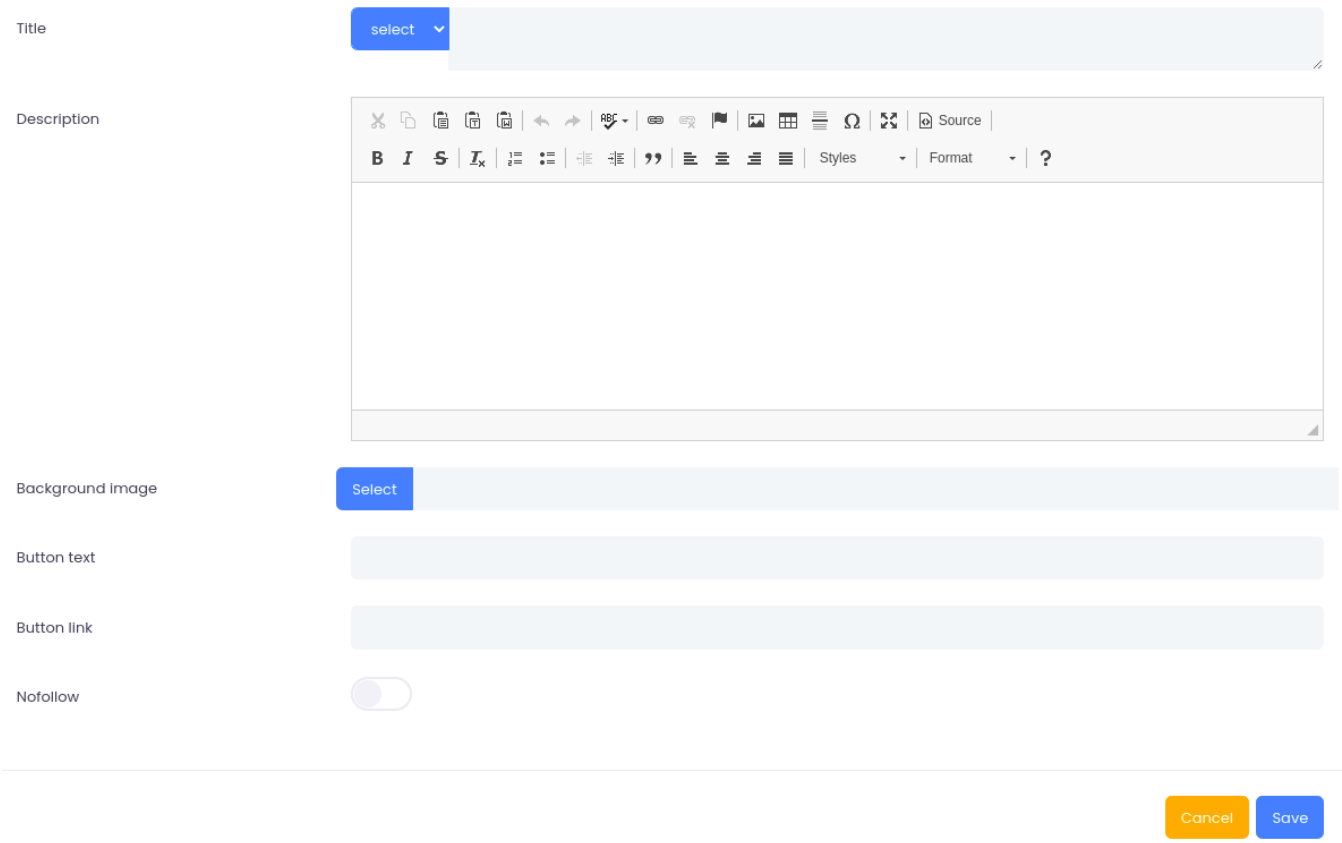


Рисунок 4.9 – Редагування даних компонента

Результатом проведених дій є веб-сторінка зображена на рисунку 4.10.

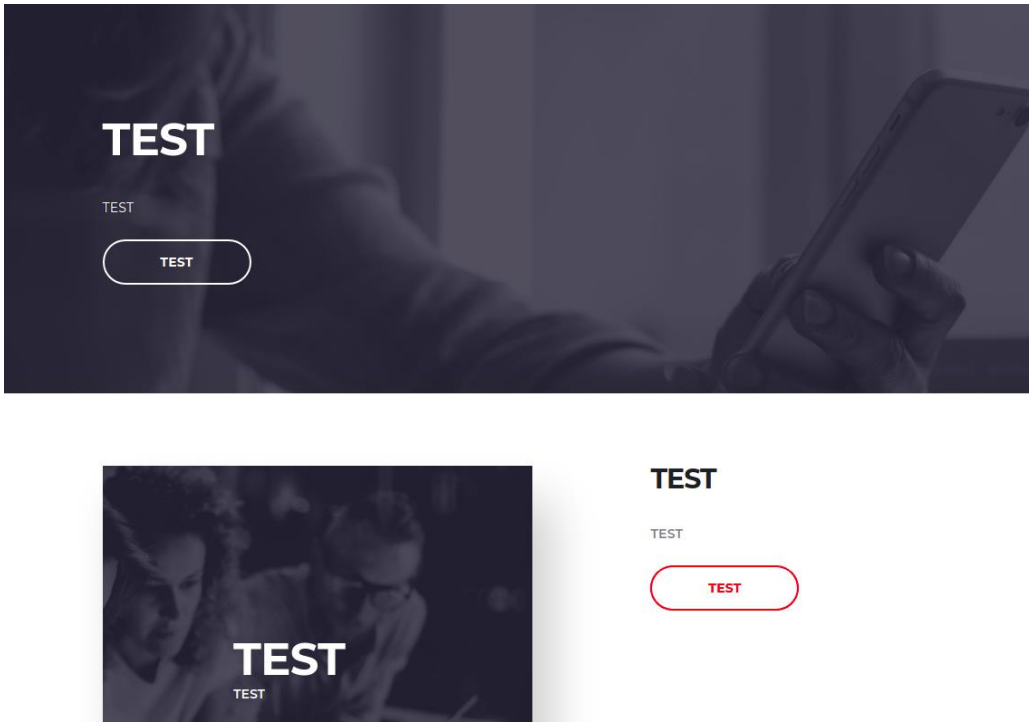


Рисунок 4.10 – Створена веб-сторінка

Наступним важливим функціоналом є робота із навігацією на сайті. Для цього необхідно перейти у розділ “navigation”, після чого з’явиться сторінка зображена на рисунку 4.11. На цій сторінці зображено деревовидну структуру, в основі якої знаходяться два елементи, що відповідають за навігацію у хедері та футері. Як і у випадку з компонентами, порядок порядок навігаційних маршрутів можна змінювати з допомогою технології “drag’n’drop”, що продемонстровано на рисунку 4.12.

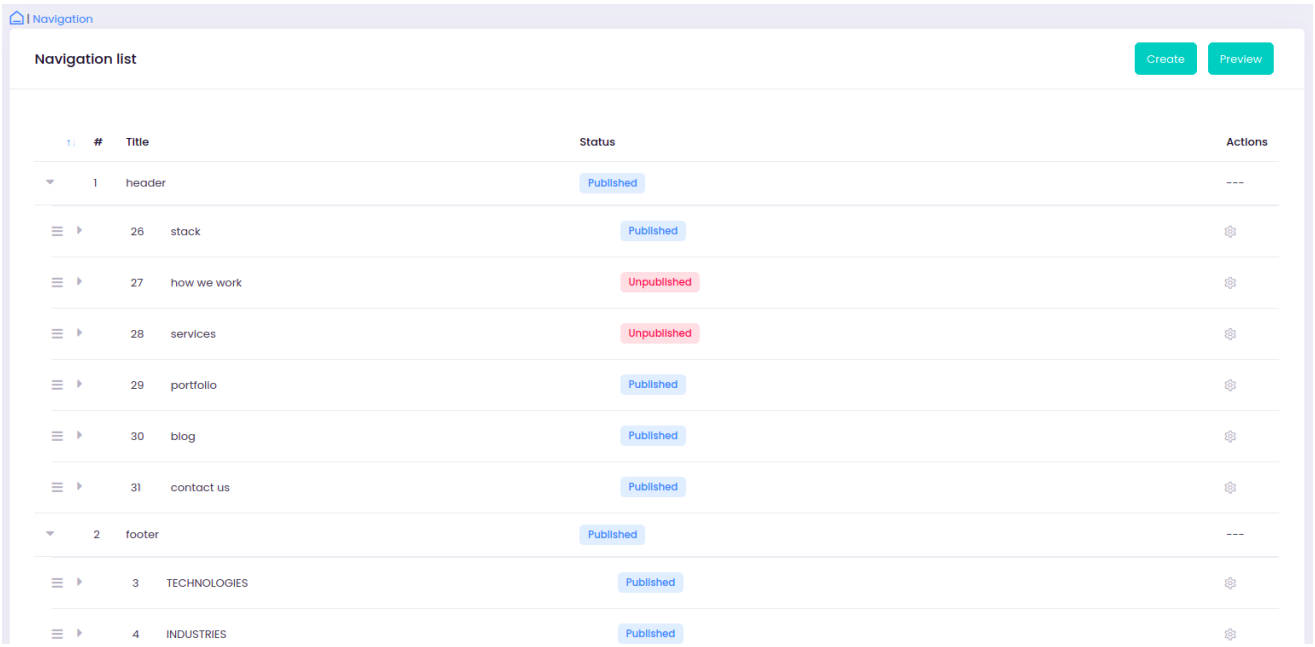


Рисунок 4.11 – Вигляд розділу “navigation”

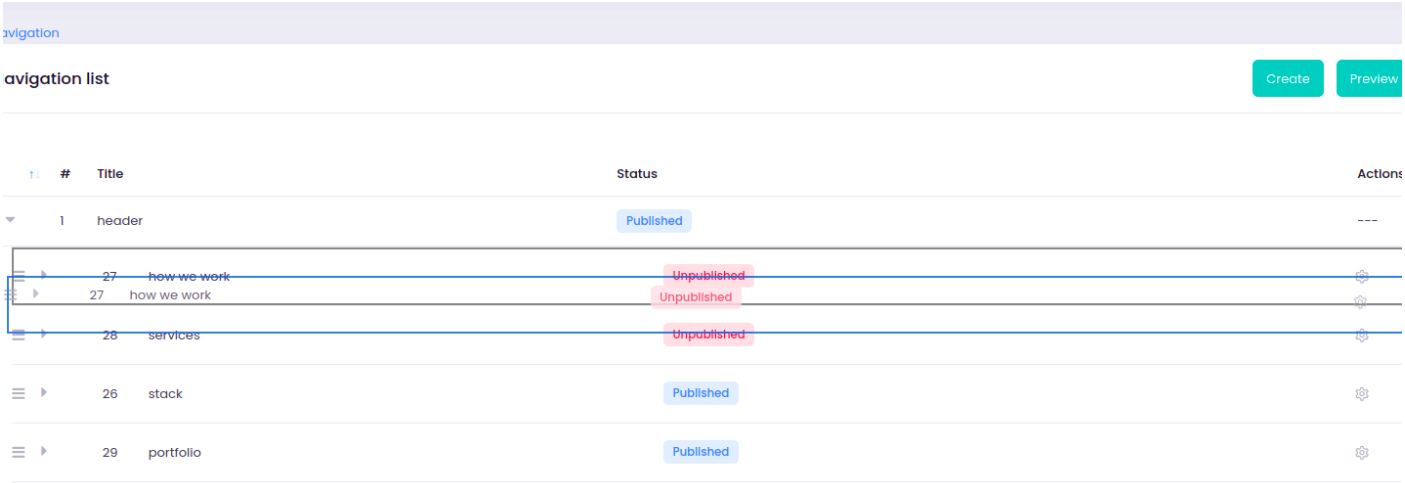


Рисунок 4.12 – Зміна порядку навігаційних маршрутів

Результат створення навігації в системі можна побачити на рисунку 4.13.

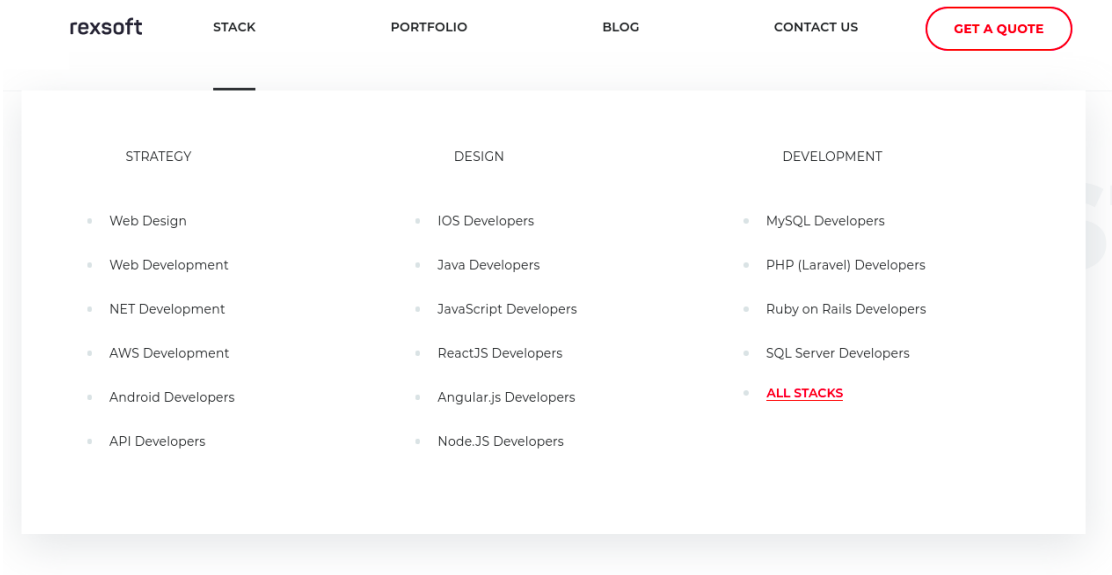


Рисунок 4.13 – Створена навігація

4.2 Перевірка системи на пошукову оптимізацію.

Для перевірки рівня пошукової оптимізації веб сторінки існують спеціальні програми. Однією із найпопулярніших таких програм є Lighthouse.

Lighthouse – це автоматизований інструмент з відкритим кодом, який призначений для покращення якості веб-сторінок. Цей інструмент є частиною веб-браузера під назвою Chrome. Для того щоб перевірити сторінки цим необхідно відкрити панель розробника кнопкою f12, потім натиснути на розділ Lighthouse. У вікні цього інструменту можна вибрати категорії для яких необхідно провести аналіз та вибрати для який типів пристроїв її проводити. Після вибору усіх необхідних критеріїв перевірки слід натиснути кнопку “Generate report” та зачекати від 30 секунд до 2 хвилин щоб отримати результати перевірки.

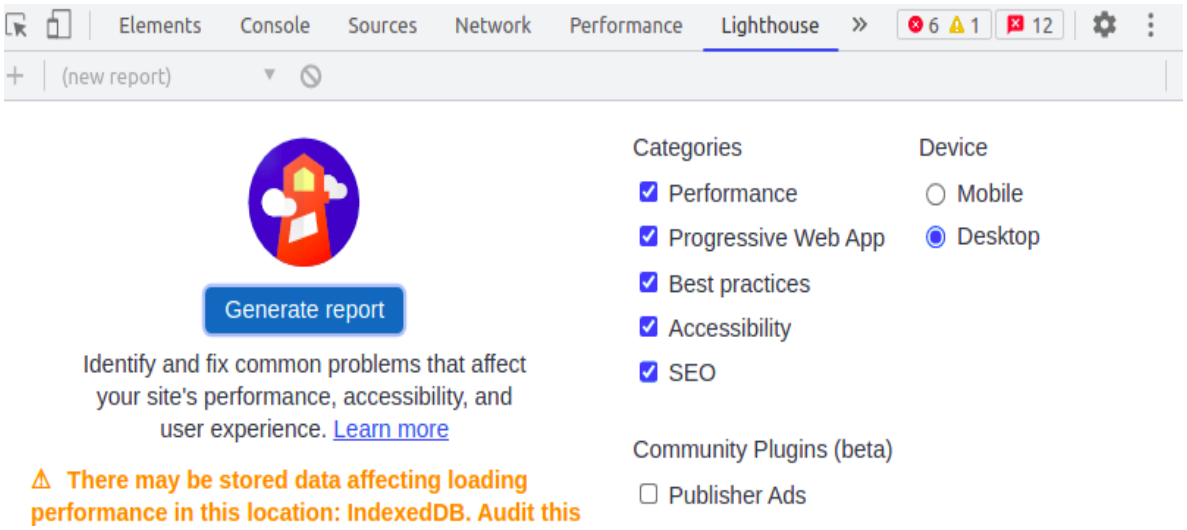


Рисунок 4.15 – Розділ Lighthouse в браузері Google Chrome.

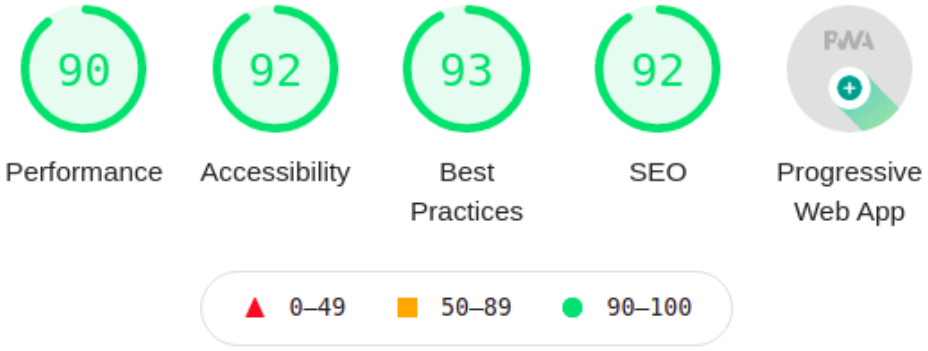


Рисунок 4.16 – Приклад перевірки сайту технологією Lighthouse.

Окрім цифр від 0 до 100 цей інструмент надає детальний звіт з кожного критерію оцінки веб сторінки. Цей інструмент також надає посилання на сторінку де описано способи вирішення помилки, яка виникла в процесі перевірки веб сторінки.

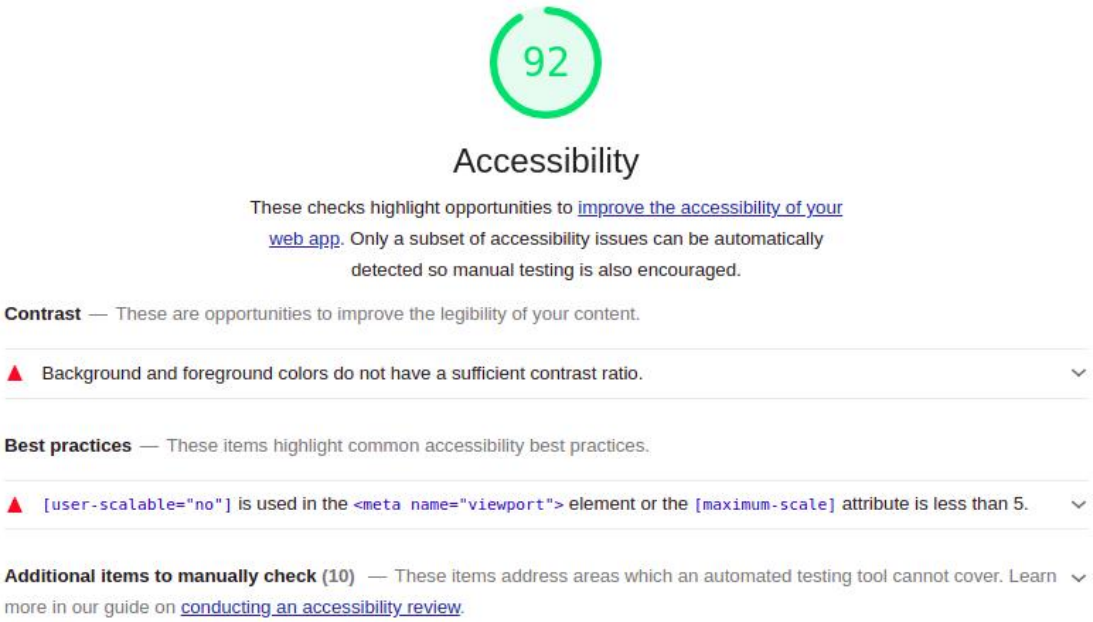


Рисунок 4.17 – Детальний опис усіх помилок допущених під час створення веб-орієнтованої системи.

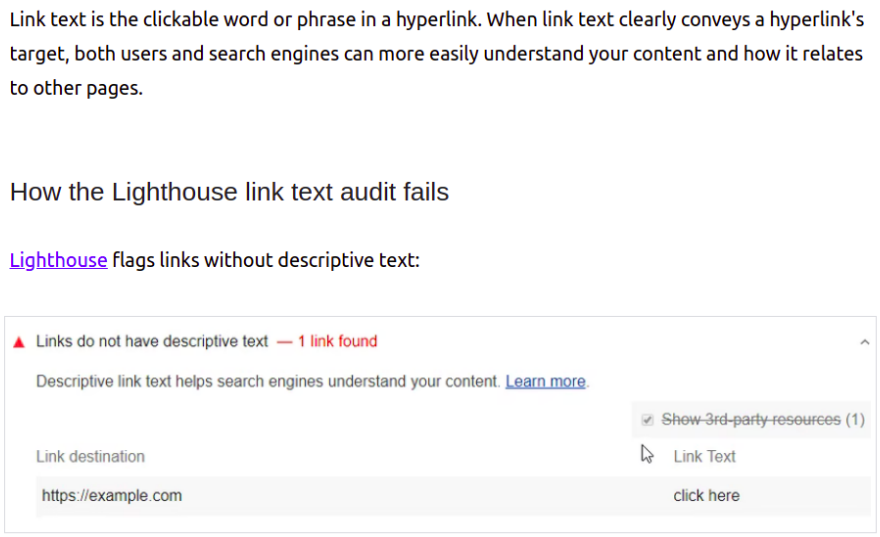


Рисунок 4.18 – Інструкція з виправлення допущеної помилки.

Отримавши детальні результати щодо відповідності сайту до вимог пошукових систем слід провести роботи на виправлення кожної з допущених помилок. Для перевірки вирішення проблем можна ще раз проаналізувати сторінку цим інструментом. Оптимальними показниками для сайту є рейтинг від 90 до 100 в кожній із представлених категорій. Такі показники гарантують, що веб сторінка буде займати високі місця в пошукових запитах.

Як видно із результату перевірки(рисунок 4.19), створена сторінка має чудові результати в порівнянні із системами-аналогами(рисунок 4.20), що можна вважати підтвердженням правильності такого підходу до створення технологій моделювання веб-орієнтованих систем.

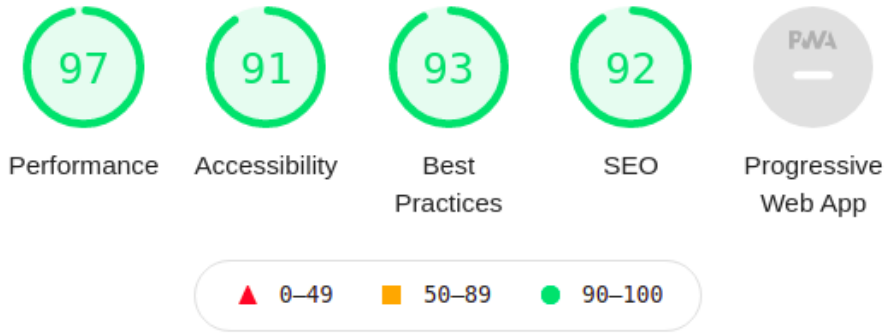


Рисунок 4.19 – Результат перевірки створеної сторінки

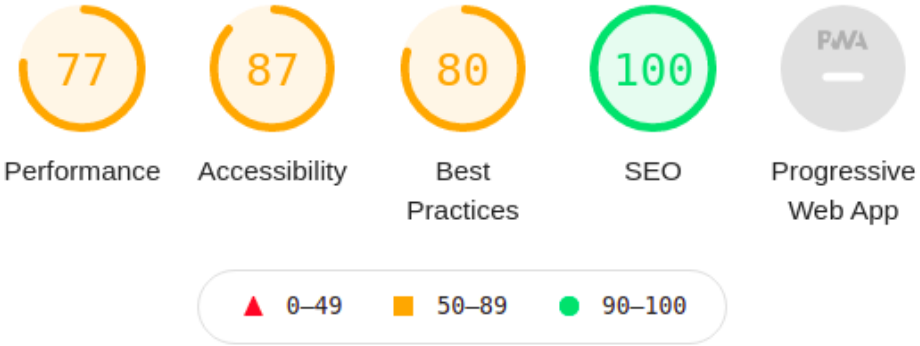


Рисунок 4.20 – Показники сайту створено за допомогою системи-аналога

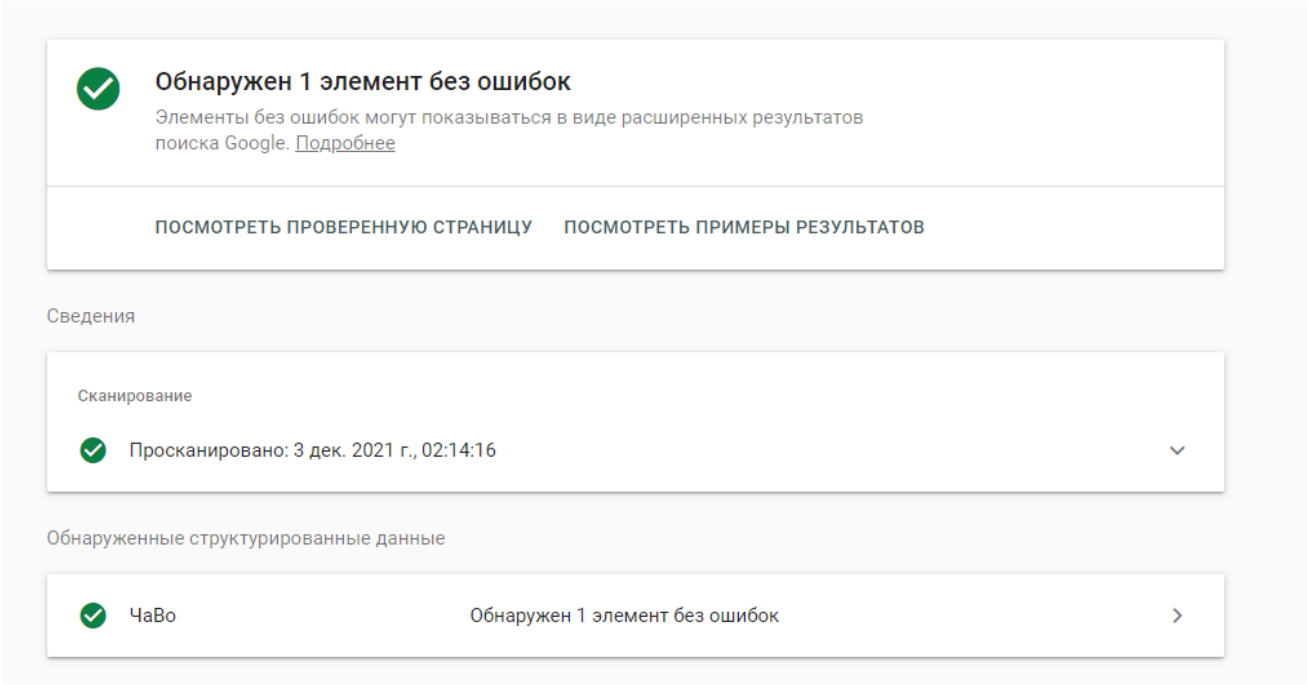


Рисунок 4.21 – Перевірка системи на використання мікророзмітки.

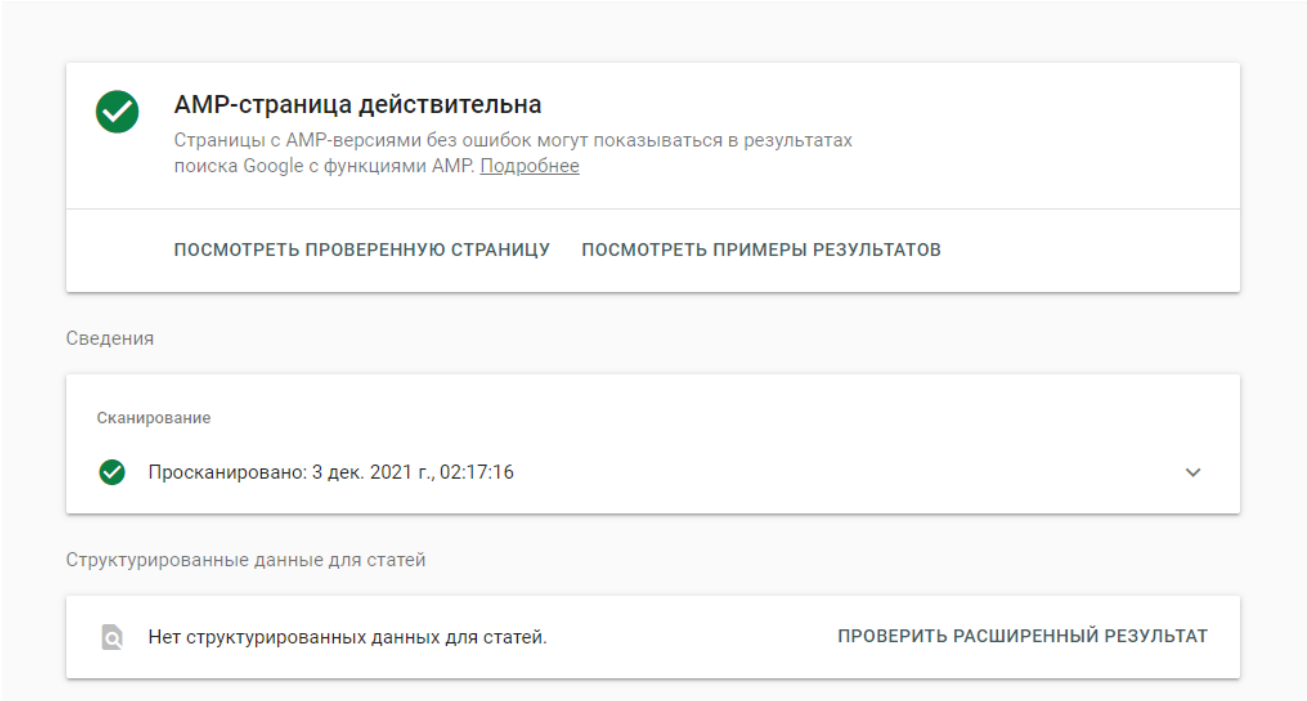


Рисунок 4.21 – Перевірка системи на відповідність усім вимогам технології AMP.

Висновки до розділу 4

У четвертому розділі був проведений детальний розгляд розробленої системи. Було описано процес створення веб-сторінок за допомогою розробленої технології. Розглянуто можливості технології для роботи із навігацією на сайті. Проведено перевірку створеної веб-сторінки на пошукову оптимізацію технологією “lighthouse”.

Висновки

В першому розділі було визначено, що таке web-орієнтована система, та було розглянуто основні їх типи. Після проведеного аналізу було обрано найпоширеніший тип такої системи та розглянуто засоби для їх моделювання. Такі засобами називаються системами контролю вмісту. Найпопулярнішими засобами моделювання є WordPress та Joomla. Такі системи є досить поширеними, але мають ряд недоліків, основними з яких є висока вразливість, складність оновлення, погана пошукова оптимізація. Як альтернативу цим засобам, було вирішено розробити власний засіб без притаманних таким системам недоліків.

В другому розділі, було описано яким чином можна удосконалити метод рендерингу динамічної сторінки на боці серверу, щоб пришвидшити швидкість роботи системи. Було запропоновано використання технологій AMP, OpenGraph та Schema.org для покращення показників системи в рейтингах пошукових систем. Також було покращено загальну схему роботи з пошуковою оптимізацією сторінки за рахунок розробленої функції, що приховує деякий контент сторінки від пошукових ботів. Удосконалення підходів до рендерингу сторінки на сервері та пошукової оптимізації сторінки є основою для створення ефективної технології для моделювання веб-орієнтованих систем.

У третьому розділі було визначено оптимальні технології для розробки інформаційної технології для моделювання веб-орієнтованих систем. До вибраного стеку технологій належать: сервер – nginx, система керування базами даних – postgresql, backend фреймворк – adonisjs. Після вибору стеку технологій було спроектовано схему бази даних, в основі якої лежить три таблиці: navigation, pages, components. Також було зображено схему взаємодії даних з кожної таблиці в процесі побудови сторінки на сервері. Далі було покроково описано процес створення функціоналу, що надає можливість створювати сторінки на модельованій системі та керувати їх контентом.

У четвертому розділі був проведений детальний розгляд розробленої системи. Був описаний спосіб потрапляння до панелі адміністратора веб-орієнтованої системи. Далі було описано весь процес створення веб-сторінок за допомогою розробленої технології. Цей процес складається із декількох основних кроків, а саме: створення сутності сторінки, додання до неї компонентів, наповнення компонентів даними. Розглянуто можливості технології для роботи із навігацією на сайті, серед яких слід відзначити можливість створення вкладеності. Проведено перевірку створеної веб-сторінки на пошукову оптимізацію технологією “lighthouse”, а також на відповідність сторінки вимогам технологій AMP та Schema.org.

Перелік посилань

1. Моделювання web-орієнтованих систем [Електронний ресурс]. – Режим доступу: http://ena.lp.edu.ua:8080/bitstream/ntb/19246/1/3-Boyko-16-25.pdf

2. Admin Dashboard [Електронний ресурс]. – Режим доступу: https://www.reviewboard.org/docs/manual/3.0/admin/admin-ui/dashboard/

3. Techniques of website speed optimization [Електронний ресурс]. – Режим доступу: https://www.altexsoft.com/blog/engineering/12-techniques-of-website-speed-optimization-performance-testing-and-improvement-practices/

4. Website conversion rate [Електронний ресурс]. – Режим доступу: https://www.geckoboard.com/best-practice/kpi-examples/website-conversion-rate/

5. Google pagespeed Insights [Електронний ресурс]. – Режим доступу: https://kinsta.com/blog/google-pagespeed-insights/

6. Lighthouse [Електронний ресурс]. – Режим доступу: https://developers.google.com/web/tools/lighthouse

7. Efficiently load javascript with defer and async [Електронний ресурс]. – Режим доступу: https://flaviocopes.com/javascript-async-defer/

8. What is minification [Електронний ресурс]. – Режим доступу: https://www.imperva.com/learn/performance/minification/

9. What is SEO? [Електронний ресурс]. – Режим доступу: https://searchengineland.com/guide/what-is-seo

10. Content relevance [Електронний ресурс]. – Режим доступу: https://www.searchmetrics.com/glossary/content-relevance/

11. Rankings [Електронний ресурс]. – Режим доступу: https://www.searchmetrics.com/glossary/rankings/

12. White hat SEO [Електронний ресурс]. – Режим доступу: https://whatis.techtarget.com/definition/white-hat-SEO

13. What is black hat SEO? [Електронний ресурс]. – Режим доступу: https://www.wordstream.com/black-hat-seo

14. What is an operating system? [Електронний ресурс]. – Режим доступу: https://edu.gcfglobal.org/en/computerbasics/understanding-operating-systems/1/

15. Web server [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server

16. Database management system [Електронний ресурс]. – Режим доступу: https://www.techopedia.com/definition/24361/database-management-systems-dbms

17. Server Side web frameworks [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks

18. Website [Електронний ресурс]. – Режим доступу: https://www.searchmetrics.com/glossary/website/

19. How a website works [Електронний ресурс]. – Режим доступу: https://www.superwebpros.com/blog/how-does-a-website-work/

20. What is a CMS [Електронний ресурс]. – Режим доступу: What is a CMS [Електронний ресурс]. – Режим доступу: https://kinsta.com/knowledgebase/content-management-system/

21. The pros and cons of using a CMS [Електронний ресурс]. – Режим доступу: https://flickerleap.com/pros-cons-using-cms-build-website/

22. Server side rendering definition [Електронний ресурс]. – Режим доступу: https://www.omnisci.com/technical-glossary/server-side-rendering

23. SEO meta tags [Електронний ресурс]. – Режим доступу: https://www.wordstream.com/meta-tags

24. The OpenGraph protocol [Електронний ресурс]. – Режим доступу: https://ogp.me/

25. Schema.org Documentation [Електронний ресурс]. – Режим доступу: https://schema.org/docs/documents.html

26. The complete guide to accelerated mobile pages [Електронний ресурс]. – Режим доступу: https://instapage.com/blog/amp

27. What is nginx [Електронний ресурс]. – Режим доступу: https://kinsta.com/knowledgebase/what-is-nginx/

28. PostgreSQL [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/

29. Adonisjs [Електронний ресурс]. – Режим доступу: https://adonisjs.com/

30. Docker [Електронний ресурс]. – Режим доступу: https://www.docker.com/

ДОДАТКИ

Додаток А

Програмні коди

```
//modules/Pages/Services/Page.js
          const View = use('View');
          const Helpers = use('Helpers');
          const fs = require('fs');
          const readFile = Helpers.promisify(fs.readFile)
          const Config = use('Config');
          const Model = use('Model');

          class PageService {
           static async render(page, additionalData) {
             await page.load('components');
             let pageJSON = page.toJSON();
             let styles = [];
             let scripts = [];
             if(pageJSON.components.length) {
               styles = pageJSON.components.reduce((acc, {type: cur}) => {
                 if(Config.get(`admin.components.${cur}.styles`)) {
                   return acc.indexOf(cur) === -1 ? acc.concat([cur]) : acc
                 } else {
                   return acc;
                 }
               }, [])
               scripts = pageJSON.components.reduce((acc, {type: cur}) => {
                 if(Config.get(`admin.components.${cur}.scripts`)) {
                   return acc.concat(Config.get(`admin.components.${cur}.scripts`)).filter((item, index) =>
acc.indexOf(item) !== index);
                 } else {
                   return acc;
                 }
               }, [])
               styles = await Promise.all(styles.map(async (el) =>
readFile(`${Helpers.appRoot()}/modules/Pages/resources/views/components/${el}/styles.css`)));
               pageJSON.components = await Promise.all(pageJSON.components.map(async ({ data, type }) => {
                 data = data || {};
                 const queries = Config.get(`admin.components.${type}.queries.view`)
                 if(queries) {
                   await Promise.all(Object.entries(queries).map(async ([key, value]) => {
                     data[key] = await value(page, data)
                     if(data[key] && data[key].rows || data[key] instanceof Model) {
                       data[key] = data[key].toJSON()
                     }
                   }))
                 }
```

```
                return View.render(`Pages.components.${type}.view`, { data, page:
pageJSON, ...additionalData}).replace(/<img(.*?)\/>/g, '<amp-img layout="intrinsic"$1></amp-img>');
              }))
            }
            styles = [
              ... await Promise.all(['general', 'grid'].map(async (el) =>
readFile(`${Helpers.appRoot()}/public/css/${el}.css`))),
              ...styles
            ]
            return { page: pageJSON, styles, scripts };
          }
        }

        module.exports = PageService;

        //modules/Pages/Controllers/ComponentsController.js
        const View = use('View');
        const Route = use('Route');
        const Database = use('Database');

        const Notify = use('ADM/Notify');
        const Datatables = use('ADM/Datatables');
        const TableBuilder = use('ADM/TableBuilder')

        const Model = use('Model');
        const Component = use('Pages/Models/Component');
        const Page = use('Pages/Models/Page');

        const Config = use('Config')

        class ComponentsController {

          async index({ view, auth, __, params, response }) {
            const { page_id } = params;

            const table = new TableBuilder('componentsTable');

            if (auth.user.managerCan('pages_create')) {
              table.setButtons([
                [`<a href="${Route.url('Pages.components.edit', { page_id })}" class="pull-right btn btn-success
no-history mx-2">${__('Pages.components.add')} ${__('Pages.components.component')}</a>`],
                [`<a href="${Route.url('Pages.pages.edit', { id: page_id })}" class="pull-right btn btn-success
no-history mx-2">${__('Pages.components.edit')} ${__('Pages.pages.page')}</a>`],
                [`<a href="${Route.url('Pages.pages.preview', { id: page_id })}" class="pull-right btn btn-success
no-history mx-2">${__('Pages.pages.preview')} ${__('Pages.pages.page')}</a>`],
              ]);
            }

            const page = await Page.find(page_id)
```

```
    table.setName(`${page.name} ${__('Pages.components.components')}`);

    table.setColums([
      { title: __('Pages.components.name'), width: '' },
      { title: __('Pages.components.type'), width: '' },
      { title: __('Pages.components.preview'), width: '' },
      { title: __('Pages.components.status'), width: '' },
      { title: __('Adm.admin.actions'), width: '1%' },
    ]);

    View.global('breadcrumbs', [
      { name: __('Adm.admin.home'), url: Route.url('admin.dashboard.index') },
      { name: __('Pages.pages.pages'), url: Route.url('Pages.pages.index') },
      { name: __('Pages.components.components'), url: Route.url('Pages.components.index', { page_id }) },
    ]);
    return view.render('Pages.component', {
      table: table.build(),
      page_id
    });
  }

  async list({ request, response, params }) {
    const { page_id } = params;

    const query = Database
      .select('components.id', 'components.name', 'components.type', 'components.published',
'components.page_id', 'components.position')
      .from('components')
      .where('components.page_id', page_id)
      .orderBy('components.position', 'asc')

    const table = new Datatables(query, request);
    const res = await table.make();
    Object.assign(res, { components: Config.get('admin.components') })
    return response.json(res);
  }

  async edit({ response, params, __, auth }) {
    const { id, page_id } = params;
    let data = await Component.find(id);
    let page = await Page.find(page_id);
    if (id && !data) {
      return response.json(Notify.error('Not found', {}));
    }

    return response.json({
      modal: {
        title: id ? __('Pages.components.edit') : __('Pages.components.create'),
        content: View.render('Pages.selectComponent', {data, page_id, page}),
        cancel: __('Adm.admin.cancel'),
```

```
            submit: __('Adm.admin.save'),
            width: 1200
          },
          success: true,
        });
      }

      async editData({ response, params, __, auth }) {
        const { id } = params;
        let component = await Component.find(id);

        const queries = Config.get(`admin.components.${component.type}.queries.form`)
        const data = {}
        if(queries) {
          await Promise.all(Object.entries(queries).map(async ([key, value]) => {
            data[key] = await value(component)
            if(data[key] && data[key].rows || data[key] instanceof Model) {
              data[key] = data[key].toJSON()
            }
          }))
        }

        if (id && !component) {
          return response.json(Notify.error('Not found', {}));
        }

        return response.json({
          modal: {
            title: id ? __('Pages.components.edit') : __('Pages.components.create'),
            content: View.render(`Pages.components.${component.type}.form`, { component, data }),
            cancel: __('Adm.admin.cancel'),
            submit: __('Adm.admin.save'),
            width: 1200
          },
          success: true,
        });
      }

      async save({ request, response }) {
        const input = request.only(['name', 'type', 'page_id', 'data']);
        input.published = request.input('published');
        const id = request.input('id');

        let component = {};
        if (!id) {
          const last = await Component.query().where('page_id', input.page_id).orderBy('position',
'desc').last();
          component = await Component.create({...input, position: last ? last.position + 1 : 1});
        } else {
          component = await Component.find(id);
```

```
      if (!component) {
        return response.json(Notify.error('Navigation not found', {}));
      }
      component.merge(input);
      if(!await component.save()) {
        return response.json(Notify.error('Not updated', {}));
      }
    }

    return response.json(Notify.success('Saved', {}));
  }

  async delete({ response, params }) {
    const { id } = params;
    const component = await Component.find(id);

    if (!component) {
      return response.json(Notify.error('Something went wrong. component not found', {}));
    }

    if (await component.delete()) {
      return response.json(Notify.success('Deleted', {}));
    }
    return true;
  }

  async reorder({request, params, response}) {
    const { page_id } = params;
    const data = request.input('data')

    let components = await Component.query().where('page_id', page_id).whereIn('position', data.map(el =>
+el.old)).fetch();
    components = components.rows;

    await Promise.all(components.map((component) => {
      const position = data.find(el => el.old == component.position).new;
      component.position = position;
      return component.save();
    }))

    return response.json(Notify.success('Saved', {}));
  }

}

module.exports = ComponentsController;

//modules/Pages/Controllers/PagesController.js
const View = use('View');
const Route = use('Route');
```

```
const Database = use('Database');

const Notify = use('ADM/Notify');
const Datatables = use('ADM/Datatables');
const TableBuilder = use('ADM/TableBuilder');

const Page = use('Pages/Models/Page');
const Component = use('Pages/Models/Component');
const Config = use('Config');
const escapeHtml = require('escape-html');
const PageService = use('Pages/Services/Page')

class PagesController {
 async index({ view, auth, __ }) {
   const table = new TableBuilder('pages');

   table.setName(__('Pages.pages.list'));

   if (auth.user.managerCan('pages_create')) {
     table.setButtons([
       [`<a href="${Route.url('Pages.pages.edit')}" class="pull-right btn btn-
success">${__('Adm.admin.create')}</a>`],
     ]);
   }

   table.setColums([
     { title: '#', width: '1%' },
     { title: __('Pages.pages.name'), width: '' },
     { title: __('Pages.pages.seo_slug'), width: '' },
     { title: __('Pages.pages.type'), width: '' },
     { title: __('Pages.pages.status'), width: '' },
     { title: __('Pages.pages.created_at'), width: '' },
     { title: __('Pages.pages.updated_at'), width: '' },
     { title: __('Adm.admin.actions'), width: '1%' },
   ]);

   View.global('breadcrumbs', [
     { name: __('Adm.admin.home'), url: Route.url('admin.dashboard.index') },
     { name: __('Pages.pages.pages'), url: Route.url('Pages.pages.list') },
   ]);

   return view.render('Pages.index', {
     table: table.build(),
   });
 }

 async list({ request, response }) {
   const query = Database
     .select('pages.id', 'pages.name', 'pages.published', 'pages.seo_slug', 'pages.type',
'pages.created_at', 'pages.updated_at')
```

```
    .from('pages')

  const table = new Datatables(query, request);
  const res = await table.make();
  return response.json(res);
}

async edit({ response, params, __, auth }) {
  const { id } = params;
  let data = await Page.find(id);
  if (id && !data) {
    return response.json(Notify.error('Not found', {}));
  }

  return response.json({
    modal: {
      title: id ? __('Pages.pages.edit') : __('Pages.pages.create'),
      content: View.render('Pages.form', { data }),
      cancel: __('Adm.admin.cancel'),
      submit: __('Adm.admin.save'),
      width: 1200
    },
    success: true,
  });
}

async save({ request, response }) {
  const input = request.only(['name', 'seo_slug', 'seo_title', 'seo_description', 'type']);
  input.published = request.input('published', false);

  const id = request.input('id');

  let page = {};

  if (!id) {
    page = await Page.create(input);
    let components = Config.get(`admin.templates.${input.type}`);
    await Promise.all(components.map((component, idx) => {
      return Component.create({
        name: component,
        type: component,
        position: idx + 1,
        page_id: page.id,
      })
    }))
  } else {
    page = await Page.find(id);
    if (!page) {
      return response.json(Notify.error('Navigation not found', {}));
    }
```

```
            page.merge(input);

            if(!await page.save()) {
              return response.json(Notify.error('Not updated', {}));
            }

          }

          return response.json(Notify.success('Saved', {}));
        }

        async delete({ response, params }) {
          const { id } = params;
          const page = await Page.find(id);

          if (!page) {
            return response.json(Notify.error('Something went wrong. page not found', {}));
          }

          if (await page.delete()) {
            return response.json(Notify.success('Deleted', {}));
          }
          return true;
        }

        async preview({ response, params, __, view }) {
          const { id } = params;
          let page = await Page.find(id);
          if (id && !page) {
            return response.json(Notify.error('Not found', {}));
          }
          const result = await PageService.render(page)

          return response.json({
            modal: {
              title: 'Preview',
              content: `
                <iframe height="700px" srcdoc="${escapeHtml(view.render('Pages.layouts.preview', result))}"
width="100%"></iframe>
              `,
              cancel: __('Pages.pages.close'),
              width: 1500
            },
            success: true,
          });
        }
      }

      module.exports = PagesController;
```

```
//modules/Pages/Validators/SaveComponentData.js
'use strict'

const Notify = use('ADM/Notify');
const Config = use('Config');

class Save {
 get rules () {
   const rules = Config.get(`admin.validationRules.${this.ctx.request.input('type')}`)
   const types = Object.keys(Config.get('admin.components')).join(',');
   return {
     id: 'required|exists:components,id',
     type: `required|in:${types}`,
     ...rules
   };
 }

 get sanitizationRules () {
   return Config.get(`admin.sanitizationRules.${this.ctx.request.input('type')}`) || {}
 }


 get validateAll () {
   return false
 }

 async fails (errorMessages) {
   return this.ctx.response.json(Notify.error(errorMessages[0].message, {}))
 }

}
module.exports = Save

//modules/Web/Controllers/WebController.js

const View = use('View');
const Page = use('Pages/Models/Page');
const Project = use('Portfolio/Models/Project');
const Category = use('Portfolio/Models/Category');
const Article = use('Articles/Models/Article');
const Route = use('Route');
const Mail = use('Mail');
const PageService = use('Pages/Services/Page')

class WebController {
 async renderPage({ params, view, response, request }) {
   const { slug } = params;
   let page = await Page.query().where('seo_slug', slug || 'home').whereIn('type', ['general', 'article',
'project']).andWhere('published', true).first();
```

```
            if(page) {
              let data = {};
              if(page.type == 'article') {
                await page.load('article');
                const article = page.getRelated('article');
                if(!article) {
                  response.status(404);
                  return view.render('Web.pages.404');
                }
                await article.loadMany(['categories'])
                const categoryIds = article.getRelated('categories').toJSON().map(el => el.id)
                const similarArticles = await Article.query().whereNot('id', article.id)
                  .whereHas('categories', (builder) => {
                    builder.whereIn('categories.id', categoryIds)
                  })
                  .withCount('categories as matches', (builder) => {
                    builder.whereIn('categories.id', categoryIds)
                  })
                  .orderBy('matches', 'desc')
                  .with('page')
                  .with('author')
                  .limit(5)
                  .fetch()
                data.previous = await Article.query().where('created_at', '>',
article.created_at).orderBy('created_at', 'asc').where('published', true).with('page').first();
                data.next = await Article.query().where('created_at', '<',
article.created_at).orderBy('created_at', 'desc').where('published', true).with('page').first();
                data.similarArticles = similarArticles.toJSON();
                const result = await PageService.render(page, data)
                return view.render('Pages.layouts.blogDetail', { ...result, ...data});
              }
              const result = await PageService.render(page, data)
              return view.render('Pages.layouts.layout', result);
            }
            response.status(404);
            return view.render('Web.pages.404');
          }

          async portfolio({ view }) {
            const projects = await Project.query().where('published', true).orderBy('created_at',
'desc').with('page').fetch();
            const categories = await Category.query().where('published', true).with('page').fetch();
            return view.render('Web.pages.portfolio.portfolio', { data: { projects: projects.toJSON(), categories:
categories.toJSON() } });
          }


          async portfolioCategory({ view, request, response, params }) {
            const { slug } = params;
```

```
            let page = await Page.query().where('seo_slug', slug).where('type', 'portfolio').andWhere('published',
true).first()
            await page.load('portfolio');
            let category = page.getRelated('portfolio')
            if(category) {
              const result = await PageService.render(page)
              return view.render('Pages.layouts.layout', result);
            }
            response.status(404);
            return view.render('Web.pages.404');
          }


          notFound({response, view}) {
            response.status(404);
            return view.render('Web.pages.404')
          }


          async contactForm({ request, response }) {
            const input = request.only(['name_form', 'name', 'email', 'text', 'checkbox', '__amp_source_origin']);

            response.header("Access-Control-Allow-Credentials", " true");
            response.header("Access-Control-Allow-Origin", "*.ampproject.org");
            response.header("AMP-Access-Control-Allow-Source-Origin", input.__amp_source_origin);
            response.header("Access-Control-Expose-Headers", "AMP-Redirect-To, AMP-Access-Control-Allow-Source-
Origin");

            try {
             const mail = await Mail.send('Pages.components.message', input, (message) => {
                message
                  .to('hello@rexsoftinc.com')
                  .from('hello@rexsoftinc.com')
                  .subject(input.name_form)
              })

              if(mail.accepted.length) {
                const redirectUrl = `${input.__amp_source_origin}${Route.url('Web.forms.thanks_page')}`;
                response.header('AMP-Redirect-To', redirectUrl)
              }
            } catch(e) {
              console.log(e);
              response.status(412).json({
                errorMessages: [
                  {
                    message: 'There is some error while sending email!',
                  }
                ]
              })
            }
```

```
            return response.json({
              successmsg: 'success'
            })
          }

          async getQuoteForm({ request, response }) {
            const input = request.only(['name_form', 'name', 'email', 'text', 'checkbox', '__amp_source_origin']);

            response.header("Access-Control-Allow-Credentials", " true");
            response.header("Access-Control-Allow-Origin", "*.ampproject.org");
            response.header("AMP-Access-Control-Allow-Source-Origin", input.__amp_source_origin);
            response.header("Access-Control-Expose-Headers", "AMP-Redirect-To, AMP-Access-Control-Allow-Source-
Origin");

            try {
             const mail = await Mail.send('Pages.components.message', input, (message) => {
                message
                  .to('hello@rexsoftinc.com')
                  .from('hello@rexsoftinc.com')
                  .subject(input.name_form)
              })

              if(mail.accepted.length) {
                const redirectUrl = `${input.__amp_source_origin}${Route.url('Web.forms.thanks_page')}`;
                response.header('AMP-Redirect-To', redirectUrl)
              }
            } catch(e) {
              response.status(412).json({
                errorMessages: [
                  {
                    message: 'There is some error while sending email!',
                  }
                ]
              })
            }

            return response.json({
              successmsg: 'success'
            })
          }
        }

        module.exports = WebController;
        //modules/Web/Middleware/isBot.js

        'use strict';

        const isbot = require('isbot');
```

```
class isbotMiddleware {
 async handle({ request, view }, next) {
   const agent = request.header('User-Agent');
   view.share({isbot: isbot(agent)})
   return next();
 }
}

module.exports = isbotMiddleware;

//modules/Pages/routes.js
const Route = use('Route');

Route.group(() => {

// Pages
Route
  .any('pages', '@provider:Pages/Controllers/PagesController.index')
  .as('Pages.pages.index')
  .middleware(['managerAuth', 'managerCan:pages_view', 'managerXHR']);
Route
  .post('pages/list', '@provider:Pages/Controllers/PagesController.list')
  .as('Pages.pages.list')
  .middleware(['managerAuth', 'managerCan:pages_view']);

Route
  .post('pages/edit/:id?', '@provider:Pages/Controllers/PagesController.edit')
  .as('Pages.pages.edit')
  .middleware(['managerAuth', 'managerCan:pages_edit']);

Route
  .post('pages/delete/:id', '@provider:Pages/Controllers/PagesController.delete')
  .as('Pages.pages.delete')
  .middleware(['managerAuth', 'managerCan:pages_delete']);

Route
  .post('pages/save', '@provider:Pages/Controllers/PagesController.save')
  .as('Pages.pages.save')
  .middleware(['managerAuth', 'managerCan:pages_edit'])
  .validator('App/../modules/Pages/Validators/Save');

 Route
 .post('pages/preview/:id', '@provider:Pages/Controllers/PagesController.preview')
 .as('Pages.pages.preview')
 .middleware(['managerAuth', 'managerCan:pages_edit']);


 //Components
 Route
 .any('pages/:page_id/components', '@provider:Pages/Controllers/ComponentsController.index')
```

```
 .as('Pages.components.index')
 .middleware(['managerAuth', 'managerCan:pages_view',  'managerXHR']);

 Route
 .post('components/list/:page_id', '@provider:Pages/Controllers/ComponentsController.list')
 .as('Pages.components.list')
 .middleware(['managerAuth', 'managerCan:pages_view']);

 Route
 .post('components/edit/:page_id/:id?', '@provider:Pages/Controllers/ComponentsController.edit')
 .as('Pages.components.edit')
 .middleware(['managerAuth', 'managerCan:pages_edit']);

 Route
 .post('components/edit-data/:id', '@provider:Pages/Controllers/ComponentsController.editData')
 .as('Pages.components.editData')
 .middleware(['managerAuth', 'managerCan:pages_edit']);

 Route
 .post('components/save', '@provider:Pages/Controllers/ComponentsController.save')
 .as('Pages.components.save')
 .middleware(['managerAuth', 'managerCan:pages_edit'])
 .validator('App/../modules/Pages/Validators/SaveComponent');

 Route
 .post('components/save-data', '@provider:Pages/Controllers/ComponentsController.save')
 .as('Pages.components.saveData')
 .middleware(['managerAuth', 'managerCan:pages_edit'])
 .validator('App/../modules/Pages/Validators/SaveComponentData');

 Route
 .post('components/delete/:id', '@provider:Pages/Controllers/ComponentsController.delete')
 .as('Pages.components.delete')
 .middleware(['managerAuth', 'managerCan:pages_delete']);


 Route
 .post('components/reorder/:page_id', '@provider:Pages/Controllers/ComponentsController.reorder')
 .as('Pages.components.reorder')
 .middleware(['managerAuth', 'managerCan:pages_edit']);


}).prefix('admin').middleware('adminPanelLocal');
```

Додаток Б

Ксерокопії наукових публікацій, виконаних при роботі над дипломною роботою магістра

Міністерство освіти і науки України
Хмельницький національний університет

ЗБІРНИК НАУКОВИХ ПРАЦЬ
за матеріалами XIII Всеукраїнської науково-практичної конференції
«Актуальні проблеми комп'ютерних наук АПКН-2021»

15-16 жовтня 2021

Хмельницький 2021

АКТУАЛЬНІ ПРОБЛЕМИ КОМП'ЮТЕРНИХ НАУК - 2021

XIII Всеукраїнська науково-практична конференція

Метою конференції є висвітлення актуальних проблем комп'ютерних наук, інформатики та інформаційних технологій.

СЕКЦІЇ КОНФЕРЕНЦІЇ:

1. Комп'ютерні науки та прикладні інформаційні технології.
2. Комп'ютерна інженерія та системи захисту інформації.
3. Математичне моделювання та інженерія програмного забезпечення
4. Телерадіокомунікації, медійні та комунікаційні системи.
5. Проблеми впровадження інформаційних технологій у виробництво та управління.

Робочі мови конференції: українська, англійська

ОРГКОМІТЕТ:

СИНЮК О. М. голова оргкомітету, проректор Хмельницького національного університету з наукової роботи, доктор технічних наук, професор

САВЕНКО О. С. заступник голови оргкомітету, декан факультету Інформаційних технологій ХНУ, доктор технічних наук, професор

БАРМАК О. В. заступник голови оргкомітету, завідувач кафедри Комп'ютерних наук ХНУ, доктор технічних наук, професор

ГОВОРУЩЕНКО Т. О. завідувач кафедри Комп'ютерної інженерії та інформаційних систем ХНУ, доктор технічних наук, професор

ВИСОЦЬКА О. В. доктор технічних наук, завідувач кафедри Радіоелектронних та біомедичних комп'ютеризованих засобів і технологій Національного аерокосмічного університету ім. М. Є. Жуковського «Харківський авіаційний інститут», професор

ЛАВРОВ Є. А. доктор технічних наук, професор (Сумський державний університет)

ТІМОФЄЄВА Л. В. відповідальна за студентську науково-дослідну роботу ХНУ

МАЗУРЕЦЬ О. В. секретар конференції, к.т.н., доцент кафедри Комп'ютерних наук ХНУ

МОЛЧАНОВА М. О. секретар конференції, викладач кафедри Комп'ютерних наук ХНУ

КОНТАКТНА ІНФОРМАЦІЯ:

e-mail для листування: *apkt.khnu@gmail.com*

УДК 004.02

Матвійчук І. І., Багрій Р. О., Скрипник Т. К.

Хмельницький національний університет

МОДЕЛЮВАННЯ WEB-ОРІЄНТОВАНИХ СИСТЕМ

Розглянуто види web-орієнтованих систем, визначено, які фактори впливають на їх ефективність. Детально досліджено процес рендерингу гіпертексту на стороні серверу та фактори, які впливають на ранжування сайту пошуковими системами. На основі проведеного дослідженння створено систему моделювання вебсайту, яка надає можливість повністю керувати його контентом.

Web-based systems are considered, and the factors influencing efficiency are determined. The process of server-side hypertext rendering and the factors that influence search engine ranking are studied in detail. Based on the study, a website modeling system was created, which allows you to fully create its content.

В теперішній час важко уявити собі людину яка ніколи не стикалася з певними інтернет ресурсами. Web-системи супроводжують людину на кожному її кроці. Вони використовуються для вирішення великої кількості завдань, та мають великий спектр застосування. Для успішної web-орієнтованої системи критично важливо мати потужну систему для контролю контенту цієї системи. Такі системи існують вже досить давно та називаються Системами керування контентом. Серед популярних систем керування вмістом слід відзначити Wordpress, Joomla, WooComerce. Ці системи мають хороший функціонал, але їх дуже важко персоналізувати, тобто додавати такий функціонал, який притаманний тільки для конкретного бізнесу.

Метою дослідження є розробка інформаційної моделювання web-орієнтованої системи.

Web-орієнтована система[1] — це будь який програмний продукт для коректної роботи якого необхідне стабільне з'єдняння з всесвітньою мережею інтернет. До web-орієнтованих систем можна віднести: сайти, мобільні додатки, настільні додатки. Розробка кожного з цих типів програмних засобів має свої унікальні особливості та потребує детального вивчення. Для розробки web-орієнтованих систем використовуються різні типи технологій та засобів. Такими засобами є: Операційні системи, Веб-сервери, бази даних, фреймворки для серверної частини системи.

Проблема керування контентом сайту актуальна завжди, навіть при наявності виликої кількості готовий рішень[3]. Всі готові рішення мають рад недоліків. Серед недоліків таких систем можна відзначити погану пошукову

оптимізацію, погану швидкість роботи та вилику кіськість вразливостей. На противагу цим великим та важким системам було розроблено невелику систему, яка буде мати великі можливості для розширення та доповнення її новими функціями. Така система повинна містити функціонал для створення нових сторінок, роботи з навігацією на сайті, роботи з даними необхідними для пошукової оптимізації сайту. Окрім якісного контенту сайт повинен мати високу швидкість завантаження, чого досить важко досягнути для повністю динамічного контенту.

Більшість web-орієнтованих систем працюють за схемою поданою на рисунку 1, проте така не повністю відповідає реальності коли мова йде про повністю динамічний контент та структуру сторінки.

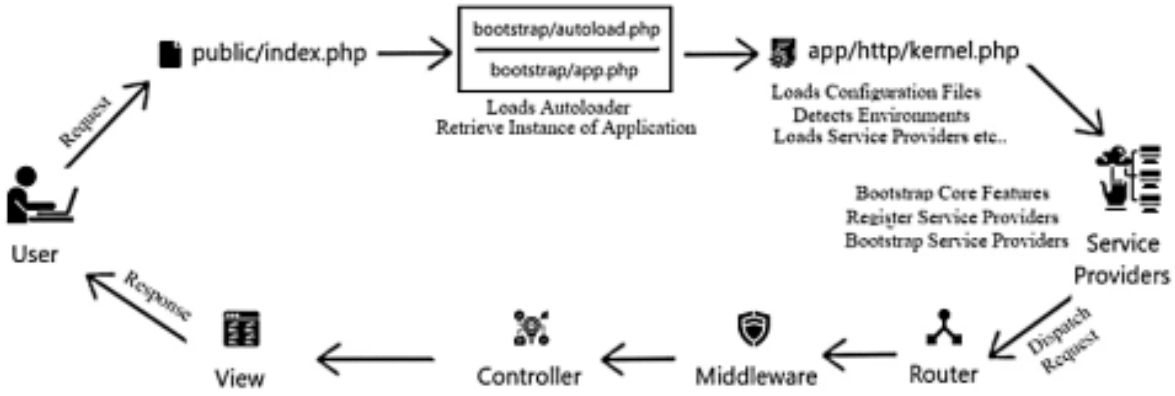


Рисунок 1– Схема роботи web-орієнтованої системи

У випадку, коли труктура сторінки вебсайту не є заздалегідь визначеною, а формується з певних налаштувань, потрібно розділити всю верстку на незалежні компоненти, які будуть мати власні стилі, скрипти, запити до бази даних. Після розбиття гіпертекстової розмітки потрібно описати функцію рендерингу, яка буде формувати всі ці компоненти в одну повноцінну та оптимізовану веб-сторінку. Такий тип рендерингу називається серверним [2] та означає здатність програми відображати веб-сторінку на сервері, а не відображати її в браузері, в результаті чого клієнту надсилається повністю відтворена сторінка.

Ще однією важливою складовою успішності web-орієнтованої системи, а саме вебсайту, є його відвідуваність.

Контент є найважливішою компонентою Веб-сайту. Якщо на ньому відсутня корисна інформація, здатна зацікавити певне коло споживачів, то ставиться під сумнів доцільність самого його функціонування. Контент є найважливішою компонентою Веб-сайту. Якщо на ньому відсутня корисна інформація, здатна зацікавити певне коло споживачів, то ставиться під сумнів доцільність самого його функціонування.

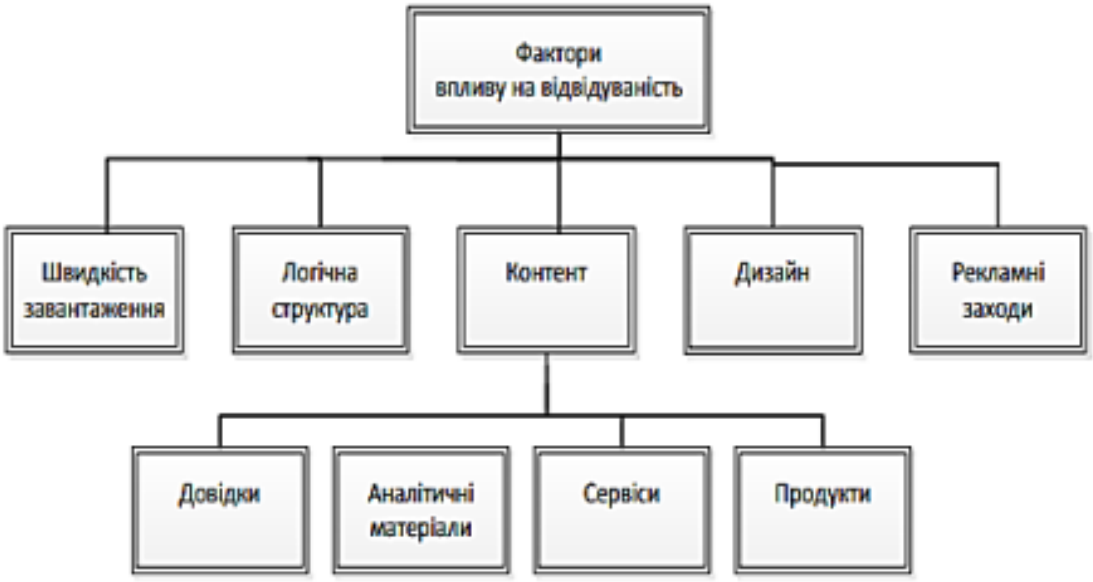


Рисунок 2 – Фактори, що впливають на відвідуваність web-орієнтованої системи

Вплинути на користувачів мережі можуть лише ті ресурси, які мають найкраще індексування (перші 10) по запитах зроблених до пошукових систем за певною темою. Ранжування матеріалів пошуковими системами здійснюється автоматично, тому існує цілий спектр правил та рекомендацій виконання яких допомає в підвищенні індексації певних матеріалів пошуковими системами, що об'єднуються одним спільним терміном SEO [4].

Під пошуковою оптимізацією (англ. search engine optimization) мають на увазі заходи для покращення позицій web-орієнтованої системи в пошукових системах по запитах відповідної цільової аудиторії. Робота з оптимізації включає насичення тексту ключовими словами із бажаних запитів користувачів, покращення оформлення тексту із застосуванням засобів виділення основних елементів, застосування прозорої навігації, оптимізація внутрішніх та зовнішніх посилань.

Окрім того необхідно забезпечити реєстрацію web-орієнтованої системи в каталогах пошукових систем, обмін посиланнями, розміщення статей в соціальних мережах, блогах, спеціалізованих Веб-сайтах. Ці методи є відомими і широко вживаними. Вони можуть сприяти хорошій відвідуваності, але не здатні її гарантувати, оскільки базуються на тимчасовій адаптації до пошукових алгоритмів конкретних пошукових систем. Інколи в гонитві за високою індексацією Веб-дизайнери подають неякісний контент та прагнуть штучно підвищувати його видимість за рахунок врахування особливостей пошукових систем, тобто із використанням методів «сірого» або «чорного» SEO.

На протидію цим підходам компанія Google в 2011 році увела до експлуатації новий алгоритм пошуку в Інтернет під назвою Panda. У квітні 2012

вийшло ще одне оновлення алгоритму Панда під назвою Пінгвін. У цьому оновленні до вимог внутрішньої якості Веб-сайту додалася ще і вимога до якості зовнішніх посилань сайту.

Рисунок 3 – Фактори впливу контенту на відвідуваність

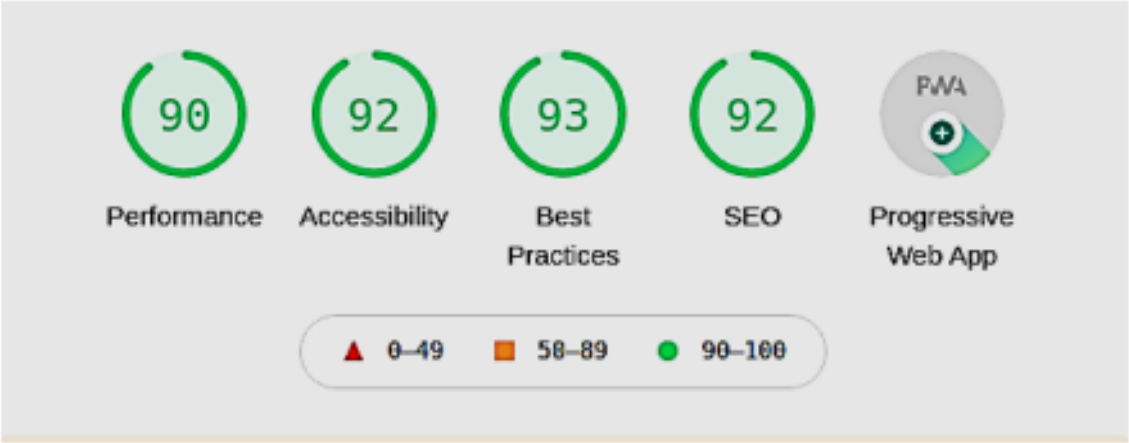


Рисунок 4 – Приклад перевірки сайту на пошукову оптимізацію

Алгоритми таких фільтрів засекречені, щоб Веб-майстри його не обійшли. Очевидно, що пошукові системи розпочали систематичну боротьбу із сірою та чорною SEO, залишаючи допустимим лише методи білої SEO, які закликають розробляти якісні сервіси для користувачів, а не для пошукових серверів. В цих умовах орієнтиром для розробників повинна стати концепція якісного наповнення

Веб-сайту. Для перевірки на пошукову оптимізацію сайту компанія Google створила інструмент під назвою Lighthouse, який є частиною веббраузеру Chrome. Приклад роботи цієї програми зображено на рисунку 4.

Цей програмний інструмент дозволяє також отримати детальну інформацію про кожну помилку, яка була допущена під час розробки чи наповнення сайту контентом (Рисунок 5).

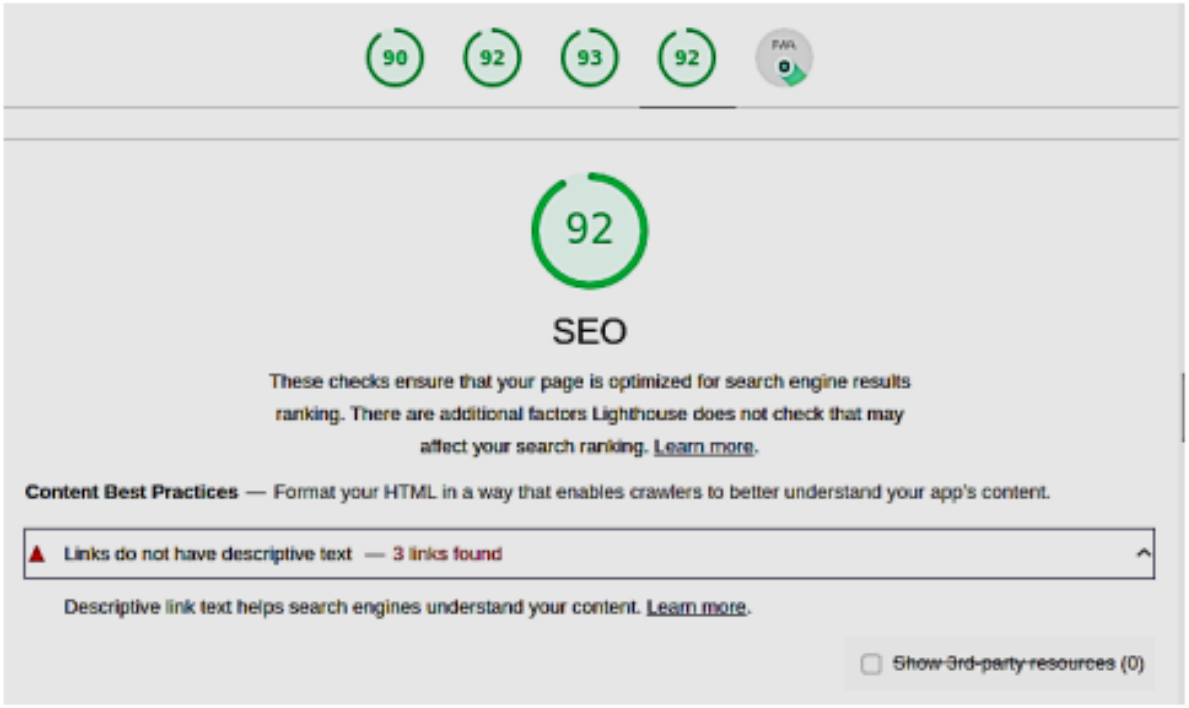


Рисунок 5 – Детальний опис допущений помилок

Отже розроблена інформаційна технологія дозволить створювати ефективні web-орієнтовані системи, а саме сайти, які будуть повністю оптимізовані та будуть добре ранжуватись у пошукових системах.

Перелік посилань

1. Web-based application [Електронний ресурс]. – Режим доступу: https://www.techopedia.com/definition/26002/web-based-application
2. Server side rendering definition [Електронний ресурс]. – Режим доступу: https://www.omnisci.com/technical-glossary/server-side-rendering
3. What is a CMS [Електронний ресурс]. – Режим доступу: What is a CMS [Електронний ресурс]. – Режим доступу: https://kinsta.com/knowledgebase/content-management-system/
4. Введение в поисковую оптимизацию [Електронний ресурс]. – Режим доступу: https://developers.google.com/search/docs/beginner/seo-starter-guide

Моделювання та адміністрування веб-орієнтованих систем

Виконав:

студент 2 курсу, групи КНм-20-1

Матвійчук І.І.

Керівник:

доцент кафедри КН

Р.О. Багрій

Мета роботи

Метою - розробка технології моделювання та адміністрування web-орієнтованих системи, яка б надала можливість налаштування контенту для покращення швидкодії побудови та пошукової індексації сторінок.

Об’єкт дослідження – процес моделювання веб-орієнтованих систем.

Предмет дослідження – моделі, методи, підходи та засоби для моделювання та адміністрування web-орієнтованих систем.

Постановка задачі

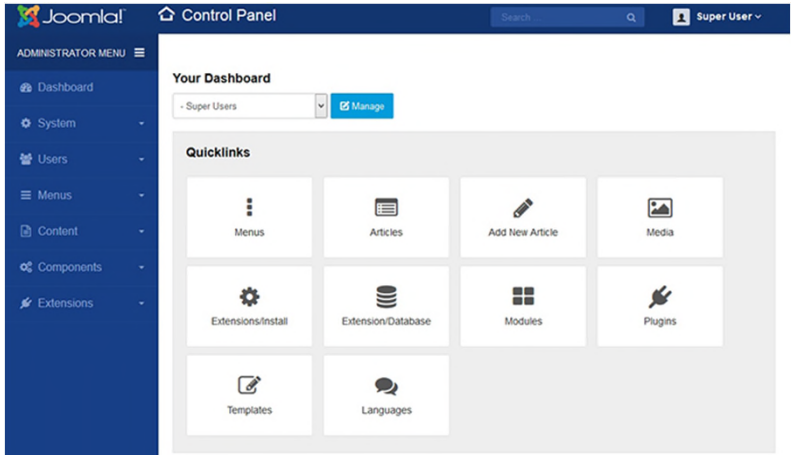
Для досягнення поставленої мети визначено наступні задачі:

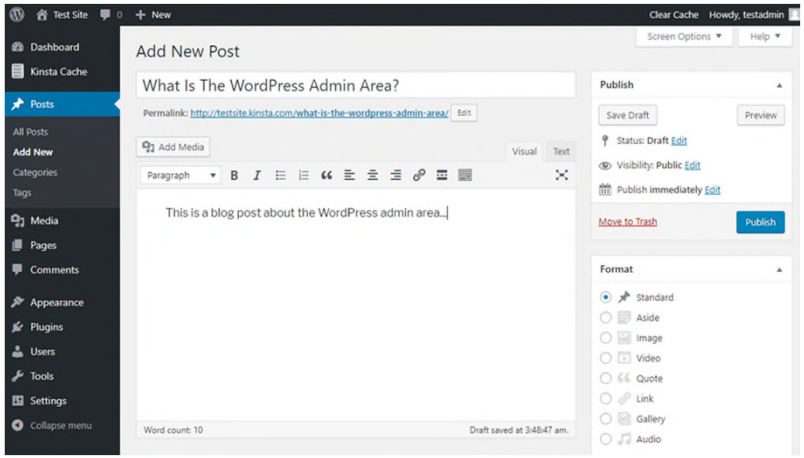
- дослідити процес побудови гіпертекстової розмітки на сервері;
- дослідити способи оптимізації web-сайтів;
- дослідити, які фактори впливають на пошукову індексацію сторінки сайту;
- спроектувати та розробити технологію моделювання та адміністрування web-орієнтованої системи з можливістю керування процесом побудови сторінки на сервері та можливістю налаштування мета-інформації, що необхідна для пошукової індексації.

Аналіз предметної області

Веб-орієнтована система — це будь який програмний продукт роботи якого необхідне з’єднання з мережею інтернет. До веб-орієнтованих систем можна віднести: сайти, мобільні додатки, настільні додатки.

Системи моделювання

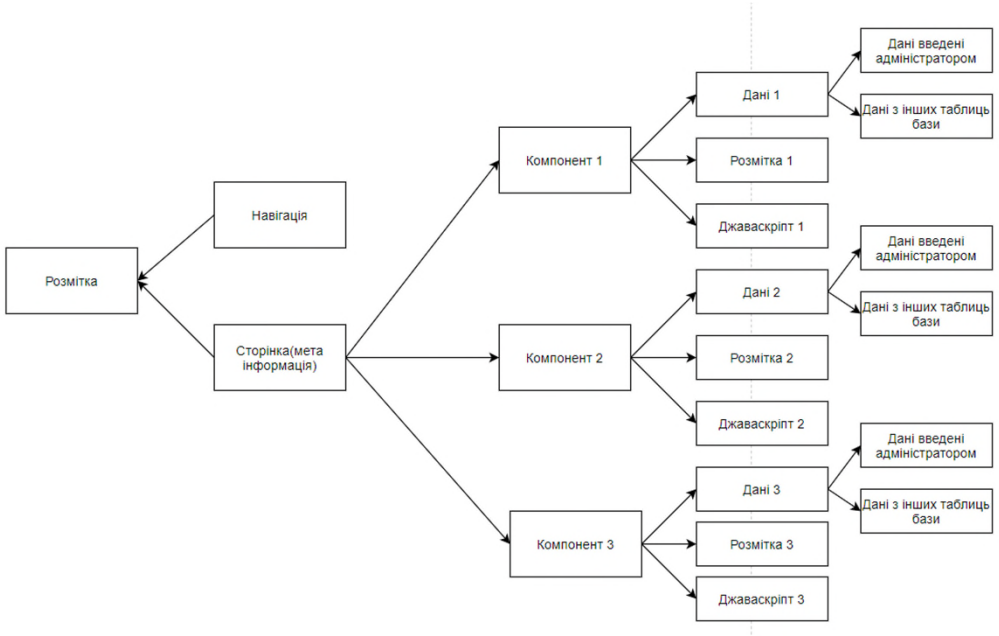




Оптимізація веб-орієнтованих систем

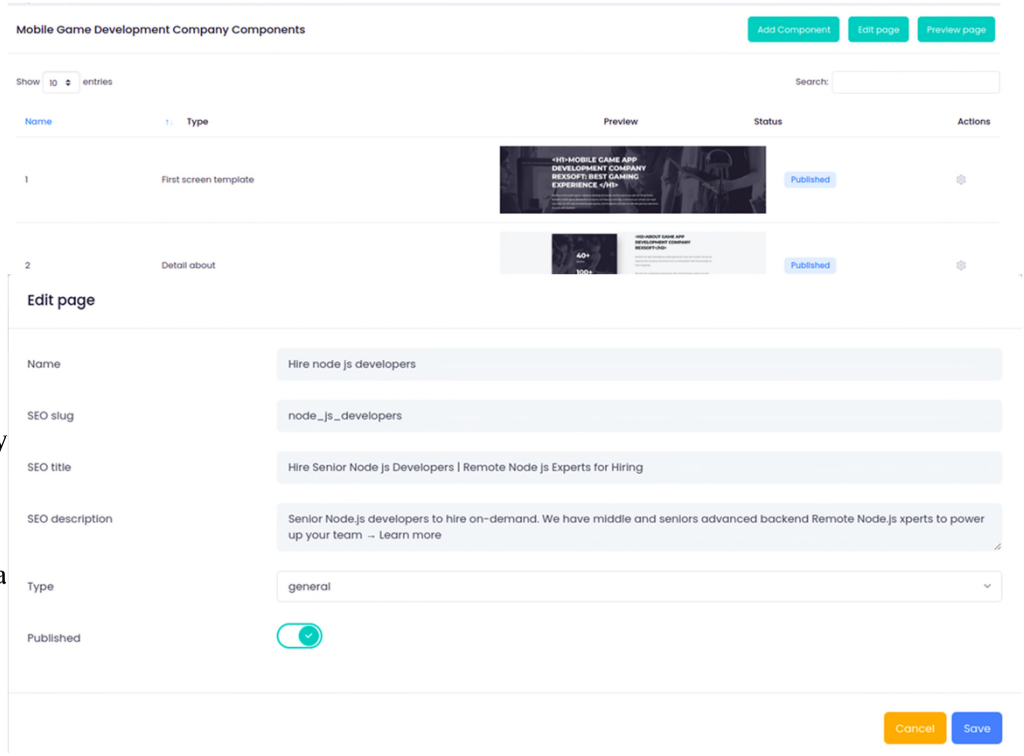
Оптимізованість — це середня кількість часу, за який веб-сторінка повністю завантажується. Швидкість завантаження розраховується від моменту, коли користувач натискає посилання або вводить URL-адресу, до моменту, коли сторінка повністю завантажується у веб-браузер, включаючи зображення, відео тощо. Швидкість завантаження залежить від таких основних факторів: веб-хостингу та сервера, коду і бази даних, які використовуються, вибору елементів дизайну, кількості і розміру зображень та відео та файлів на кожній сторінці, версії браузера, поведінки користувачів.

Метод побудови динамічної сторінки



Пошукова оптимізація

Пошукова оптимізація сторінки залежить від двох основних критеріїв: якість контенту представленого на веб-сторінці та рівня технічної реалізації. До контенту, що впливає на пошукову оптимізацію відноситься мета-інформація та текст, що відображається в тілі сторінки. У панелі адміністратора, що була створена, пропонується повне керування контентом як самої сторінки, так і її мета-інформації.



Наукова новизна одержаних результатів

1. вдосконалено алгоритм рендерингу динамічного контенту веб-сторінки, що дає можливість підвищити швидкість роботи веб-орієнтованої системи;
2. вдосконалено метод пошукової оптимізація веб-сайту, що за допомогою визначеного набору інструментів дозволив покращити показники індексації веб-орієнтованої системи;
3. розроблено технологію моделювання та адміністрування web-орієнтованої системи з можливістю керування процесом побудови сторінки на сервері та налаштуванням мета-інформації, що необхідна для пошукової індексації.

Практичне значення одержаних результатів

На основі розробленої технології моделювання та адміністрування веб-систем створено програмну реалізацію у вигляді панелі адміністратора з можливістю керування процесом побудови сторінки та налаштуванням мета-інформації, що дає можливість оптимізувати процес пошукової індексації.

Висновки

Визначено, що таке веб-орієнтована система, та які існують типи таких систем. Проведено аналіз існуючих технологій моделювання веб-систем. Визначено основні фактори, що впливають на якість системи. Розроблено власний метод побудови веб-сторінки на сервері. Розроблено панель адміністратора веб-орієнтованої системи з можливістю керування процесом побудови сторінки на сервері та можливістю налаштування мета-інформації, що необхідна для пошукової індексації.

Mon Nov 29 15:48:12 EET 2021, Петровський Сергій Степанович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 9.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 13%**

	Документ		Суммарное совпадение по Базе Данных	
ID: 97513 Название: Моделювання та адміністрування веб-орієнтованих систем Добавлено в БД: 2021-11-29 Авторы: І.І. Матвійчук Руководители: Р.О. Багрій Консультанты: Опоненты:	Символы	Лексемы	Символы	Лексемы
	66447	580	9732 (15%)	77 (13%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы

UNICHECK

Ім'я користувача:
Кафедра КН

ID перевірки:
1009410007

Дата перевірки:
29.11.2021 16:44:13 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
29.11.2021 16:44:50 EET

ID користувача:
100005671

Назва документа: **Матвійчук_магістерська_Lite**

Кількість сторінок: **74** Кількість слів: **9996** Кількість символів: **80404** Розмір файлу: **4.06 MB** ID файлу: **1009427929**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

8.8%
Схожість

Найбільша схожість: **4.16%** з Інтернет-джерелом (https://victana.lviv.ua/metodychky/137-tekhnolohii-elektronnoi-kome..

5.36% Джерела з Інтернету	24	Сторінка 76
4.09% Джерела з Бібліотеки	76	Сторінка 76

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	5
Підозріле форматування	13 сторінок

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНИХ НАУК

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ МАГІСТРА ДО ЗАХИСТУ ЗА РЕЗУЛЬТАТАМИ АНАЛІЗУ ЗВІТУ ПОДІБНОСТІ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: *Моделювання та адміністрування веб-орієнтованих систем*

Автор: *Матвійчук Іван Ігорович*

Спеціальність: 122 – Компютерні науки

Освітня програма: освітньо-професійна

Науковий керівник: к.т.н., доц. Багрій Р.О.

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) за програмою Anti-Plagiarism виявлені 9% є фрагментарними – містять поширені конструкції, загальновідомі терміни, скорочення та визначення.
2) За програмою UNICHECK виявлені 8.68%, що є запозиченнями, які розміщені в розділах аналізу існуючих технологій та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 9% і 8.68% відповідно, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Руслан Багрій

Гарант ОП _____ Руслан Багрій

Завідувач кафедри КН _____ Олександр Бармак

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МОН УКРАЇНИ
кафедра комп'ютерних наук

ВІДГУК ОПОНЕНТА

на кваліфікаційну роботу магістра

гр. КНм-20-1 Матвійчука Івана Ігоровича за темою: *Моделювання та адміністрування веб-орієнтованих систем*

1. **Актуальність обраної теми**

Тема адміністрування та моделювання веб-орієнтованих систем є актуальною незважаючи на велику кількість існуючих інструментів, що було добре відображено у кваліфікаційній роботі магістра.

2. **Відповідність роботи предметній області спеціальності 122 Комп'ютерні науки та загальним вимогам до наукових робіт**

Робота відповідає вимогам предметної області спеціальності 122 Комп'ютерні науки, оскільки в роботі виконуються теоретичні та експериментальні дослідження в галузі комп'ютерних наук, застосовуються алгоритмічні принципи в моделюванні, проектуванні, розробці та супроводі інформаційних систем.

3. **Повнота розкриття мети та завдань дослідження**

Метою магістерської роботи є розробка технології моделювання та адміністрування web-орієнтованих системи, яка б надала можливість налаштування контенту для покращення швидкодії побудови та пошукової індексації сторінок. Робота повністю розкриває поставлену мету та усі завдання, що були поставлені.

4. **Наявність наукової новизни**

В роботі описуються загальновідомі методи вирішення конкретних проблем веб-орієнтованих систем, студентові вдалося застосувати їх для побудови динамічних сторінок на боці серверу. Для цього студент спроектував власний метод рендерингу гіпертекстової розмітки. Окрім цього в роботі було проведено дослідження факторів та способів, що впливають на пошукову оптимізацію системи. Зважаючи на це можна зробити висновок, що в роботі наявна наукова новизна в достатньому обсязі. Також слід відзначити, що проведене дослідження було опубліковано на XIII всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2021» 15 листопада 2021 р., м. Хмельницький, Україна.

5. **Зміст кожного розділу роботи**

В першому розділі подано визначення web-орієнтованої системи, розглянуто основні їх типи. Проведено аналіз найпоширеніших типів систем та розглянуто засоби для їх

моделювання. Визначено недоліки систем контролю вмісту та поставлено задачу розробити власний засіб з врахуванням цих недоліків.

В другому розділі вдосконалено метод рендерингу сторінки на боці серверу, що пришвидшує швидкість роботи системи. Також покращено загальну схему роботи з пошуковою оптимізацією сторінки за рахунок розробленої функції, що приховує деякий контент сторінки від пошукових ботів. Удосконалення підходів до рендерингу сторінки на сервері та пошукової оптимізації сторінки є основою для створення ефективної технології для моделювання веб-орієнтованих систем.

У третьому розділі визначено оптимальні технології для розробки інформаційної технології для моделювання веб-орієнтованих систем. Після вибору технологій спроектовано схему бази даних, покроково описано процес створення функціоналу, що надає можливість створювати сторінки у системі та керувати їх контентом.

У четвертому розділі проведено детальний розгляд розробленої системи. Описано процес створення веб-сторінок за допомогою розробленої технології. Розглянуто можливості технології для роботи із навігацією на сайті. Проведено перевірку створеної веб-сторінки на пошукову оптимізацію технологією “lighthouse”.

6. Ступінь розкриття теми роботи

В роботі недостатньо уваги приділено адмініструванню веб-орієнтованих систем. Основний акцент роботи був спрямований на моделювання веб-орієнтованих систем та розробку методу побудови веб-сторінки на боці серверу.

7. Якість оформлення кваліфікаційної роботи

Загалом оформлення роботи відповідає поставленим вимогам, але є певні недоліки що стосуються мови та граматики.

8. Недоліки кваліфікаційної роботи

В роботі недостатньо уваги приділено адмініструванню веб-орієнтованих систем, також є несуттєві недоліки в оформлені.

9. Загальний висновок (допускається чи не допускається до захисту), якої оцінки заслуговує кваліфікаційна робота.

Роботі виконана студентом в достатньому обсязі, наявна наукова новизна та виконані усі поставлені завдання. Кваліфікаційна робота допускається до захисту, але враховуючи допущені помилки, заслуговує оцінку “добре”.

Опонент Ольга Павлова
д.ф., ст.викл. кафедри КІС

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МОН УКРАЇНИ
кафедра комп'ютерних наук

ВІДГУК НАУКОВОГО КЕРІВНИКА

на кваліфікаційну роботу магістра

гр. КНм-20-1 Матвійчука Івана Ігоровича за темою: Моделювання та адміністрування веб-орієнтованих систем

1. Актуальність теми

Актуальність теми достатньо обґрунтована, оскільки моделювання та адміністрування веб-орієнтованих систем надає можливість налаштування контенту для покращення швидкодії побудови та пошукової індексації сторінок.

2. Відповідність роботи предметній області спеціальності 122 Комп'ютерні науки та загальним вимогам до наукових робіт

Теми кваліфікаційної роботи "Моделювання та адміністрування веб-орієнтованих систем" відповідає предметній області спеціальності 122 Комп'ютерні науки та вимогам до кваліфікаційної роботи магістра, оскільки об'єктом дослідження є процес моделювання веб-орієнтованих систем, предметом дослідження – моделі, методи, підходи та засоби для моделювання та адміністрування web-орієнтованих систем.

3. Професійні та особистісні якості магістранта

Матвійчук І. І. під час роботи над кваліфікаційною роботою магістра продемонстрував достатній рівень знань та умінь за спеціальністю "Комп'ютерні науки".

4. Ступінь самостійності під час виконання кваліфікаційної роботи

Робота виконана самостійно, академічного плагіату не виявлено, стосовно всіх запозичень наведено відповідні посилання на джерела.

5. Наукова новизна та оригінальність запропонованих підходів

Отримані такі результати: вдосконалено алгоритм рендерингу динамічного контенту веб-сторінки, що дає можливість підвищити швидкість роботи веб-орієнтованої системи; вдосконалено метод пошукової оптимізація веб-сайту, що за допомогою визначеного набору інструментів дозволив покращити показники індексації веб-орієнтованої системи; розроблено технологію моделювання та адміністрування web-орієнтованої системи з можливістю керування процесом побудови сторінки на сервері та налаштуванням мета-інформації, що необхідна для пошукової індексації. Отримані результати оприлюднені на XIII всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних

наук АПКН-2021» 15 листопада 2021 р., м. Хмельницький, Україна, доповідь на тему «Моделювання web-орієнтованих систем».

6. Ступінь оволодіння методами дослідження

Студент Матвійчук І. І. має достатній ступінь володіння методами дослідження, що були використанні у роботі.

7. Повнота та якість розкриття теми роботи

Мета роботи повністю розкрита, отримані результати підтверджують достовірність наукових положень.

8. Логічність, послідовність, аргументованість, літературна грамотність викладу матеріалу

Викладення матеріалу логічне, послідовне та аргументоване. Мова і стиль викладення кваліфікаційної роботи магістра відповідають стандартам, що забезпечує доступність сприймання матеріалу і відповідає вимогам до сучасних наукових робіт.

9. Можливість практичного застосування кваліфікаційної роботи, окремих її частин

Може мати практичне значення при адмініструванні веб-орієнтованих систем.

10. Висновок про можливість допуску кваліфікаційної роботи до захисту, на яку оцінку заслуговує робота

Вважаю, що кваліфікаційна робота студента Матвійчука Івана Ігоровича може бути рекомендована до захисту та заслуговує на оцінку "добре".

Науковий керівник _____ к.т.н., доц. Руслан Багрій