Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

МОДЕЛЮВАННЯ ТА РОЗРОБКА ПЛАТФОРМИ ДЛЯ СТВОРЕННЯ ВЕБСТОРІНОК
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-19-2

Коротков В.О.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Машталір В.П.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Короткову Владиславу Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Моделювання та розробка платформи для створення вебсторінок

затверджена наказом університету від 15 травня 2023 року № 474 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 27 травня 2023 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, мова програмування JavaScript, фреймворк Solid.js, фреймворк Nest.js, база даних PostgreSQL, середовище розробки WebStorm.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз існуючих платформ для створення вебсторінок.
2. Моделювання архітектури платформи для створення вебсторінок.
3. Розробка платформи для створення вебсторінок.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми створення вебсайтів, постановка задачі, платформа для створення вебсторінок.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Консультант з дотримання діючих стандартів та норм	Доцент Творошенко І.С.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	10.04.2023	
2	Аналіз завдання, підбір літератури	11.04.23-17.04.23	
3	Аналіз літератури з досліджуваної проблеми	18.04.23-20.04.23	
4	Аналіз предметної області	21.04.23-30.04.23	
5	Постановка задачі	01.05.23-14.05.23	
6	Програмна реалізація	15.05.23-23.05.23	
7	Оформлення пояснювальної записки	24.05.23-26.05.23	
8	Перевірка на плагіат	27.05.23	
9	Рецензування	28.05.23	
10	Підготовка презентації та доповіді	29.05.23-04.06.23	
11	Занесення роботи в електронний архів	05.06.23	
12	Попередній захист кваліфікаційної роботи	05.06.23	

Дата видачі завдання 10 квітня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ ____проф. Машталір В.П._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 67 с., 4 табл., 24 рис., 41 джерело.

ВЕБЗАСТОСУНКИ, КОНСТРУКТОРИ САЙТІВ, АРХІТЕКТУРА, ВЕБСАЙТ, ВЕБДИЗАЙН, СИСТЕМА, ПЛАТФОРМА, БАЗА ДАНИХ, FRONTEND, BACKEND.

Об’єктом роботи є моделювання та розробка платформи для створення вебсторінок.

Метою роботи є розробка мінімального життєздатного продукту (MVP) платформи для створення вебсторінок. За допомогою технології необхідно реалізувати зручну платформу, де можна буде легко та інтуїтивно створювати вебсторінки різного направлення.

Використано методи розробки проєктів у IT-галузі, технології frontend та backend розробки, спосіб авторизації через зберігання даних користувача на сервері, спосіб автоматичної публікації вебсторінок на піддомені платформи.

У результаті роботи був проведений аналіз предметної галузі, виявлено проблемні місця та поставлено задачу розробки готового рішення. Були сформовані вимоги та реалізовано дизайн платформи та досліджено методи розробки вебсайтів

WEB APPLICATIONS, WEBSITE BUILDERS, ARCHITECTURE, WEBSITE, WEB DESIGN, SYSTEM, PLATFORM, DATABASE, FRONTEND, BACKEND.

The object of the work is the modeling and development of a platform for creating web pages.

The aim of the work is to develop a minimum viable product (MVP) for a web page creation platform. With the help of modern programming languages, frameworks and architecture patterns, it is necessary to implement a convenient platform where it will be possible to easily and intuitively create web pages of various directions.

The methods of project development in the IT industry, frontend and backend development technologies, a method of authorization through storing user data on the server, a method of automatic publication of web pages on the platform subdomain were used.

As a result of the work, the analyses of subject area was made, problem areas and set the task of developing a ready-made solution were identified. The requirements were formed and the platform design was implemented, and the methods of website development were investigated.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмний застосунок

БД – база даних

ПК – персональний комп’ютер

СКБД – система керування базами даних

HTML – Hyper Text Markup Language (мова гіпертекстової розмітки)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

JS – Java Script

SEO – Search Engine Optimization (пошукова оптимізація)

AJAX – Asynchronous JavaScript and XML

DOM – Document Object Model (об’єктна модель документа)

CSSOM – CSS Object Model (об’єктна модель каскадних таблиць стилів)

MVP − Minimum Viable Product (мінімально життєздатний проєкт)

API − Application Programming Interface (інтерфейс програмування застосунків)

AI − Artificial Intelligence (штучний інтелект)

ВСТУП

Вебтехнології відіграли найважливішу роль в історії розвитку сучасного інформаційного суспільства. Після появи перших сторінок у 1990-х роках йшов постійний процес вдосконалення та розширення вебресурсів. Тепер стало можливим використовувати вебсторінки як основний інструмент для обміну інформацією та надання послуг у всьому світі.

У ранній стадії розвитку вебтехнологій створення сайтів здійснювалося за допомогою HTML та CSS. Згодом до них долучилися такі технології, як JavaScript, який дозволив реалізувати інтерактивність, а також PHP, Python та інші мови програмування, що дозволяють створювати динамічні вебсайти. На сьогоднішній день основні технології, які використовуються для розробки вебсторінок, включають HTML, CSS, JavaScript та різноманітні фреймворки та бібліотеки [1].

Однак створення вебсайту з нуля може бути трудомістким процесом, який вимагає від розробників глибоких знань та навичок у програмуванні та дизайні. У зв’язку з цим, на ринку з’явилися платформи та конструктори вебсторінок, такі як Tilda, Wix та Webflow. Вони дозволяють користувачам без спеціальних навичок і знань створювати якісні вебсайти [2].

Такі конструктори сайтів мають ряд переваг, зокрема забезпечують швидкість та простоту розробки, а також знижують вартість створення сайтів. Це робить їх особливо привабливими для малого та середнього бізнесу, оскільки їм не потрібно витрачати значні ресурси на залучення професійних розробників та дизайнерів.

У той же час, незважаючи на існування таких платформ, ринок конструкторів вебсторінок продовжує розвиватися і зростати. Це зумовлено тим, що нові технології та тенденції у сфері вебдизайну та програмування вимагають постійного оновлення можливостей існуючих платформ. Окрім цього, різні бізнеси та користувачі висувають унікальні вимоги до функціональності та дизайну своїх вебсайтів, тому широкий спектр

конструкторів лише сприяє задоволенню потреб різноманітних груп користувачів.

Таким чином, актуальність проблеми полягає в необхідності створення нових платформ для розробки вебсторінок, які б відповідали сучасним технологічним вимогам та надавали користувачам можливість легко та швидко створювати вебсайти з різними функціональними можливостями та дизайном.

1 АНАЛІЗ ІСНУЮЧИХ ПЛАТФОРМ ДЛЯ СТВОРЕННЯ ВЕБСТОРІНОК

1.1 Сучасний стан розвитку платформ для створення вебсторінок

Розвиток сучасних технологій надає безліч можливостей для запуску та розвитку бізнесу. Відзначається неймовірний ріст кількості інтернет-магазинів у сучасному світі. Але з правильним маркетингом та просуванням, можна з успіхом створити свій власний онлайн-магазин. У зв'язку з підвищенням попиту на створення інтернет-магазинів, сформувалися спеціальні платформи – конструктори вебсайтів. За допомогою таких платформ можна легко створити власний сайт-візитку, односторінковий сайт або повноцінний онлайн-магазин без необхідності глибоких знань програмування або дизайну: конструктори дозволяють зібрати сайт як пазл.

Конструктор вебсайтів – це програмне забезпечення, яке дозволяє створювати сайти в візуальному редакторі без спеціальних знань програмування. Вони можуть бути надані як окрема послуга або як додаток до хостингових пакетів. Безкоштовні плани зазвичай надають базовий хостинг та піддомен.

За допомогою конструкторів вебсайтів можна створити свій власний сайт за короткий час без знань HTML, CSS, JavaScript; легко змінити дизайн та структуру сайту; керувати контентом, таким як текст, зображення, відео тощо, та оптимізувати сайт для пошукових систем без спеціальних знань SEO.

Конструктори вебсайтів поділяються на онлайн-конструктори та програми-конструктори. Онлайн-конструктори працюють безпосередньо у браузері, а дані зберігаються на серверах провайдера. Зазвичай, послуги зберігання є платними, і користувачі обмежені тарифними планами. Програми-конструктори встановлюються на комп'ютері як додаток. Офлайн-конструктори вебсайтів працюють на зразок графічних редакторів, проте на виході ви отримуєте архів із сторінками майбутнього сайту. Цей архів можна

завантажити на будь-який хостинг, придбати доменне ім’я та опублікувати в інтернеті.

Використання конструкторів вебсайтів дозволяє швидко розробити та легко оновлювати вебресурси, адаптуючи їх до змінних потреб бізнесу та споживачів. Завдяки інтуїтивному інтерфейсу та готовим шаблонам, користувачі можуть працювати з конструкторами без попередньої підготовки [3].

Однак, слід враховувати, що готові конструктори можуть мати обмеження щодо функціональності, індивідуального дизайну та можливостей оптимізації для пошукових систем. Відповідно, великі компанії або бізнеси зі складними технічними вимогами можуть віддавати перевагу вебсайтам розробленими професійними розробниками.

У світлі цього, конструктори вебсайтів найбільше підходять для малого та середнього бізнесу, фрілансерів, особистих проєктів та освітніх ресурсів. Вони значно спрощують процес створення сайту, знижують вартість розробки та забезпечують доступ до онлайн-присутності для широкого кола користувачів.

У подальшому розвитку конструкторів вебсайтів слід зосередитись на поліпшенні гнучкості, індивідуалізації та оптимізації вебсайтів, а також на розширенні можливостей підтримки та навчання користувачів. Враховуючи різноманітність потреб ринку та вимог споживачів, наступні покоління конструкторів вебсайтів можуть стати ще потужнішими та доступнішими для широкого загалу користувачів.

Серед сучасних трендів у розвитку конструкторів вебсайтів можна виділити забезпечення більшої інтеграції з іншими онлайн-сервісами, такими як CRM-системи, маркетплейси, соціальні мережі, платіжні системи та інші. Це дозволить користувачам створювати єдину екосистему для свого бізнесу, спрощуючи керування та підвищуючи ефективність [4].

Також слід звернути увагу на розвиток мобільних версій конструкторів вебсайтів, які дозволять користувачам створювати та редагувати вебсайти

безпосередньо зі своїх смартфонів або планшетів. Це забезпечить зручність роботи та зниження залежності від комп’ютерів.

У контексті навчання веброзробці, конструктори вебсайтів, такі як WebFlow, можуть стати відмінним засобом для опанування основ програмування та дизайну. Платформи надають доступ до візуального редактора, який дозволяє користувачам відчути роботу з кодом, а також ресурсів для вивчення технічних аспектів створення сайту. Таким чином, конструктори вебсайтів можуть сприяти розвитку навичок веброзробки та підготовці нових фахівців у цій галузі.

Враховуючи перераховані аспекти, конструктори вебсайтів продовжують розвиватися та вдосконалюватися, надаючи все більше можливостей для користувачів різного рівня та потреб. Незалежно від розміру або складності бізнесу, такі інструменти можуть допомогти забезпечити швидке створення та розширення онлайн-присутності, сприяючи успіху підприємств та індивідуальних проєктів.

Адаптивний дизайн та економія ресурсів стають ключовими вимогами до сучасних вебсайтів. Конструктори вебсайтів, які реагують на ці потреби, розширюють свій асортимент інструментів, що дозволяють створювати вебсайти, оптимізовані для різних пристроїв та платформ. Це забезпечує підвищення зручності користувачів, а також поліпшення показників продуктивності та ефективності сайту [5].

Виробники конструкторів вебсайтів вже активно інтегрують штучний інтелект та машинне навчання у свої продукти. Це сприяє автоматизації процесів, таких як аналіз даних, оптимізація структури сайту, адаптація дизайну під різні екранні розміри, та генерація контенту. У результаті користувачі отримують ще більше можливостей для створення високоякісних, індивідуальних та конкурентоспроможних вебсайтів з мінімальними зусиллями та ресурсами.

З огляду на все вище сказане, конструктори вебсайтів продовжують розвиватися, відповідаючи на зростаючі потреби ринку та споживачів. Вони стають все більш гнучкими, функціональними та доступними, дозволяючи користувачам створювати вебсайти з різними ступенями складності та бюджетів. Враховуючи ці фактори, конструктори вебсайтів можуть стати ключовим інструментом для майбутнього розвитку онлайн-підприємництва та індивідуальних проєктів.

1.2 Аналіз схожих платформ

На сьогоднішній день існує велика кількість платформ і конструкторів вебсторінок, які надають користувачам можливість легко та швидко створювати вебсайти. Серед них виділяються такі популярні платформи, як Tilda [6], Wix [7] та Webflow [8]. Вони різняться за функціональністю, спрямованістю та цільовою аудиторією.

Tilda – це інтуїтивний конструктор сайтів, орієнтований на створення стильних та візуально привабливих вебсайтів. Він ідеально підходить для створення продаючих сторінок, візиток, блогів та інших невеликих проєктів. Основні переваги Tilda полягають в простоті використання, широкому виборі шаблонів, адаптивному дизайну та інтеграції з різними сервісами.

Tilda відрізняється від інших конструкторів сайтів своїм фокусом на візуальному дизайні та гнучкості у налаштуванні візуального зовнішнього вигляду. Основний принцип роботи з Tilda полягає в використанні готових блоків, які можна об’єднати та перетягувати у візуальному редакторі (рис. 1.1). Кожен блок має набір параметрів, які дозволяють користувачам налаштувати зовнішній вигляд, анімацію та поведінку блоків на сторінці (рис. 1.2).

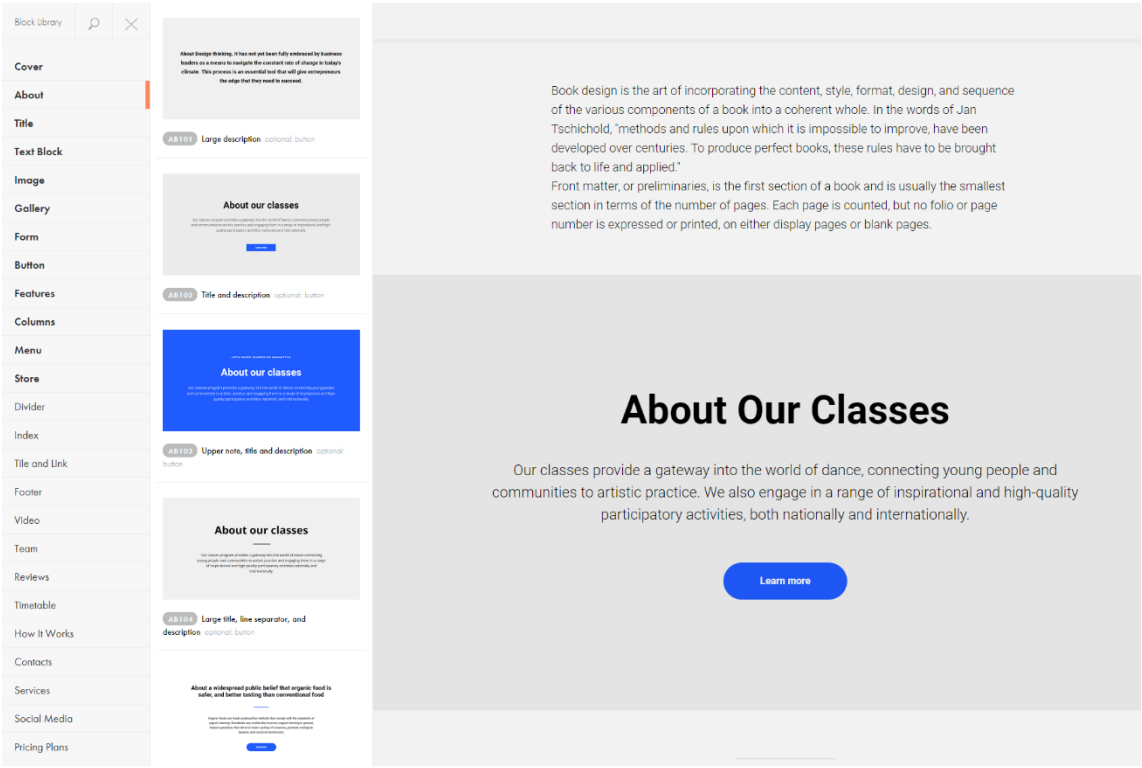

Рисунок 1.1 – Візуальний редактор сторінки Tilda

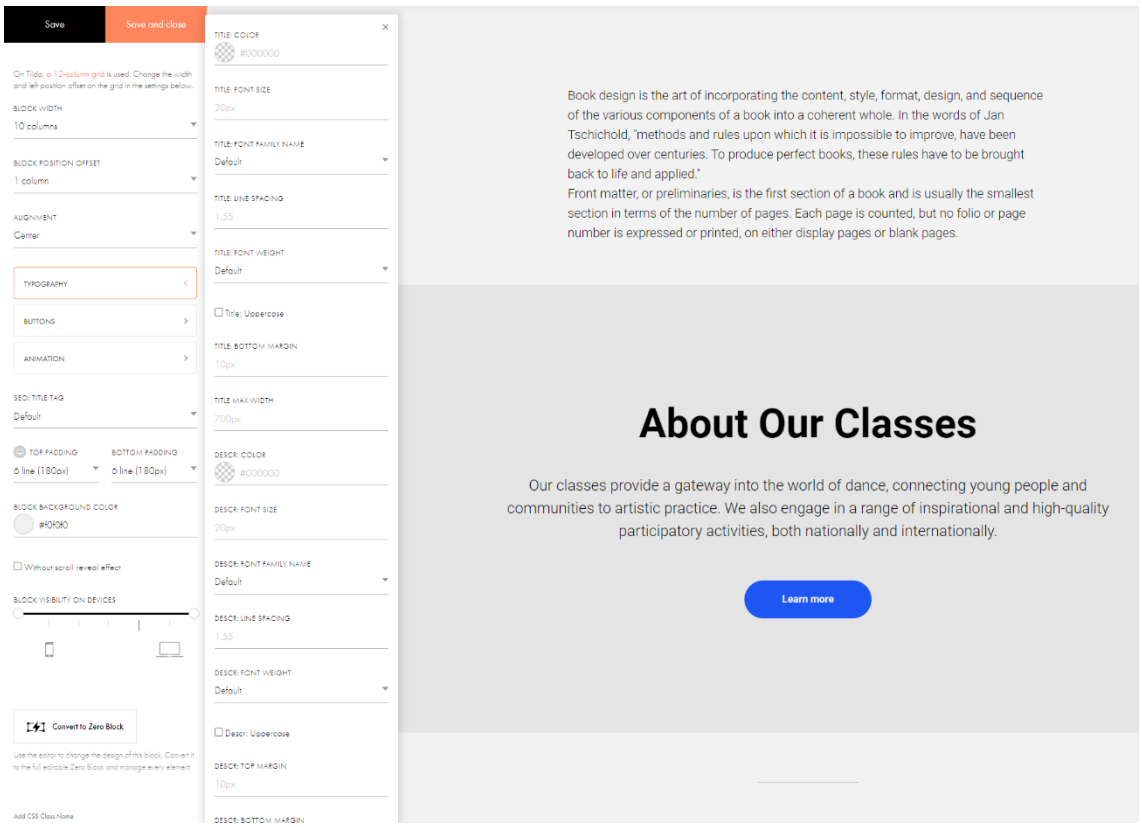

Рисунок 1.2 – Візуальний редактор блоку Tilda

Цільовою аудиторією Tilda є малі та середні підприємства, фрілансери, блогери, а також некомерційні організації, які прагнуть створити візуально привабливі та професійно виглядаючі вебсайти без значних фінансових та часових витрат.

Wix – одна з найбільш відомих платформ для створення вебсайтів, яка пропонує велику кількість шаблонів та функцій для різних типів вебпроєктів. Це універсальний конструктор сайтів, підходить як для особистих, так і для корпоративних сайтів, блогів, інтернет-магазинів тощо. Wix має потужний візуальний редактор, а також можливість розширення функціоналу за допомогою плагінів.

Wix пропонує інтуїтивний досвід користувача, завдяки своєму візуальному редактору, який базується на принципі «drag-and-drop» (рис. 1.3). Це дозволяє користувачам легко переміщати та змінювати елементи сторінки без будь-яких технічних знань. Велика кількість готових шаблонів, які можуть бути адаптовані до потреб користувача, робить Wix привабливим для широкого кола користувачів.

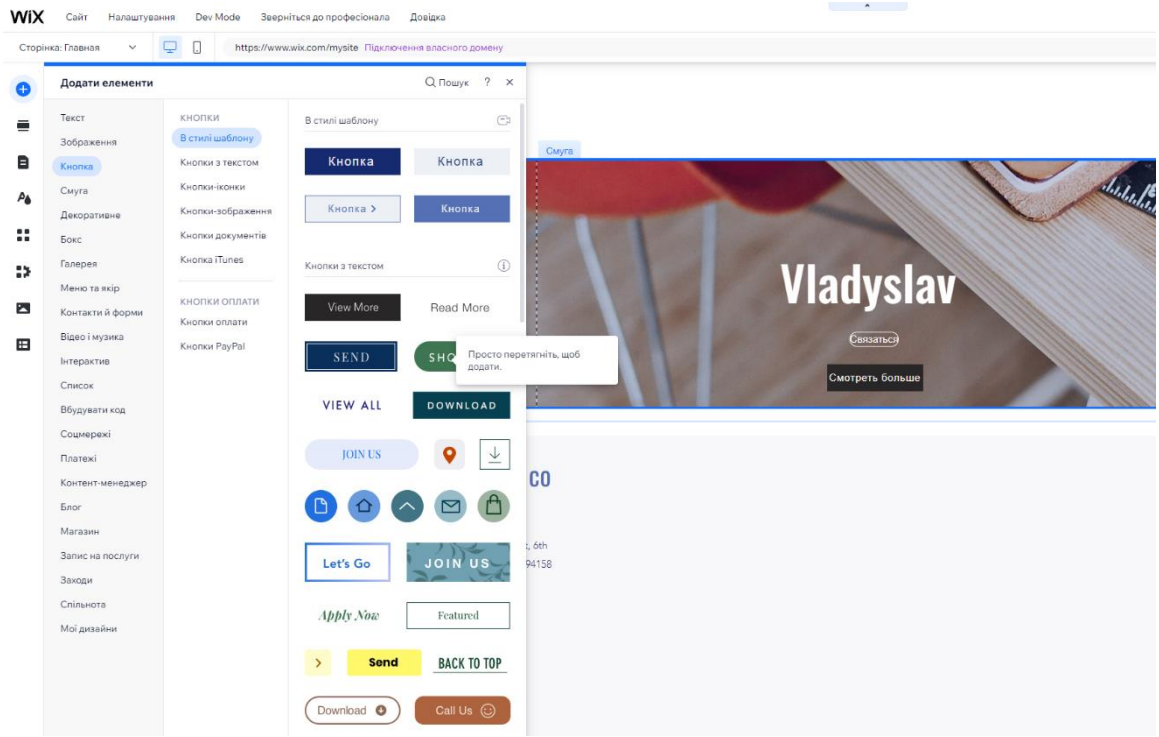

Рисунок 1.3 – Візуальний редактор сторінки Wix

Однією з основних фіч Wix є його App Market, який містить сотні плагінів та додатків, що допомагають розширити функціональність вашого вебсайту. Від соціальних медіа інтеграцій до систем управління клієнтами, Wix App Market пропонує рішення для більшості потреб власників вебсайтів.

Цільовою аудиторією Wix є фрілансери, малі та середні підприємства, а також особи, які прагнуть створити власний блог або особистий сайт. Завдяки широкому спектру функціональних можливостей та гнучкості у налаштуваннях, Wix забезпечує зручний та надійний спосіб створення вебсайтів для різних типів проєктів.

Однією з переваг Wix є його оптимізація для пошукових систем, яка допомагає сайтам досягти вищих позицій у результатах пошуку. Інтегровані SEO-інструменти та рекомендації забезпечують користувачам можливість налаштовувати ключові слова, метатеги та інші параметри, що сприяють покращенню видимості вебсайту.

Wix також пропонує хмарне зберігання та хостинг, що полегшує процес розгортання та управління вебсайтом. Користувачі можуть легко зареєструвати доменне ім’я та забезпечити швидке завантаження сторінок завдяки глобальній мережі доставки контенту (CDN), яку надає Wix.

У підсумку, Wix є потужним та гнучким конструктором сайтів, який підходить як для початківців, так і для досвідчених розробників. Завдяки широкому спектру інструментів, шаблонів та розширень, Wix забезпечує користувачам можливість створити власні вебсайти, що відповідають їх вимогам та потребам.

Webflow – це більш продвинута платформа, яка поєднує в собі можливості конструктора сайтів та CMS. Він надає користувачам гнучкість в налаштуванні дизайну, структури сайту та функціоналу, а також можливість працювати з HTML, CSS та JavaScript на вищому рівні.

Webflow надає відмінний візуальний редактор, який дозволяє користувачам вільно маніпулювати елементами сторінки, змінюючи стилі, шрифти та кольори за допомогою інтуїтивно зрозумілих

інструментів (рис. 1.4). Він також пропонує широкий вибір готових шаблонів, які можна налаштувати за потребами конкретного проєкту. Завдяки підтримці CSS Grid та Flexbox, користувачі можуть легко створювати адаптивні макети, які відповідають сучасним стандартам вебдизайну.

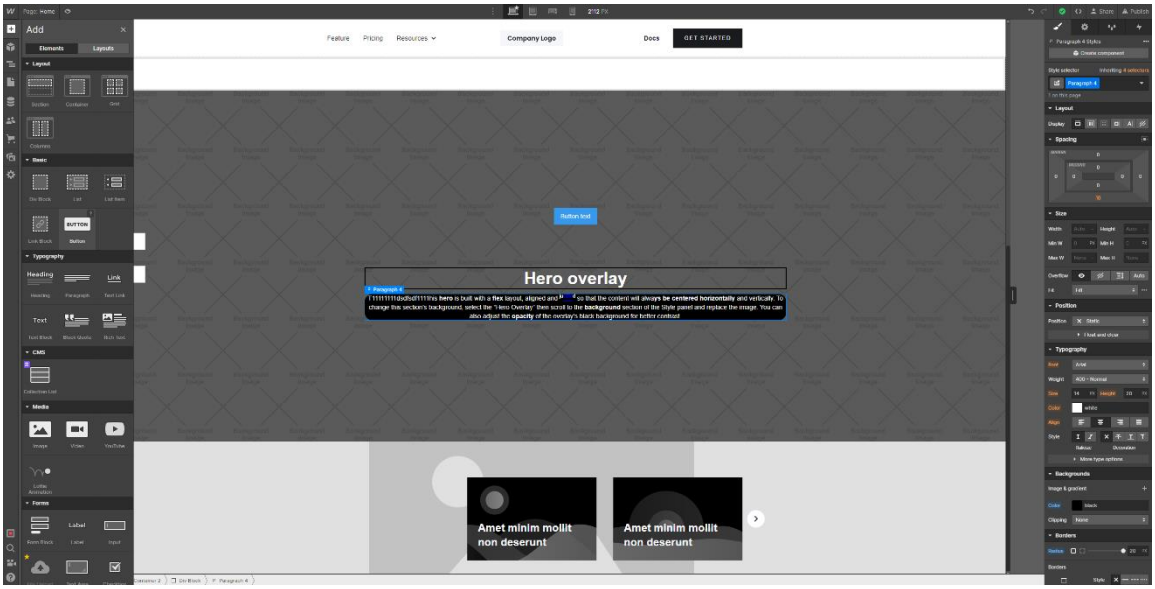

Рисунок 1.4 – Візуальний редактор сторінки WebFlow

Система керування контентом (CMS) Webflow дозволяє користувачам створювати та управляти динамічним контентом, таким як блоги, галереї, списки продуктів та інші типи вебресурсів. Завдяки інтеграції з API, можна підключати сторонні сервіси для розширення функціоналу вебсайту або зберігання даних. Також Webflow пропонує можливість створення та управління інтернет-магазинами, включаючи налаштування каталогів, кошиків, систем оплати та доставки.

Цільова аудиторія Webflow – це, в основному, вебдизайнери та розробники, які хочуть мати повний контроль над своїми проєктами і віддають перевагу ручному налаштуванню коду, а не автоматичним конструкторам. Тим не менш, завдяки інтуїтивному інтерфейсу та доступним навчальним ресурсам, Webflow також може бути корисним для менш досвідчених користувачів, які хочуть розвивати свої навички у вебдизайні та розробці.

Розглянувши схожі платформи, можна виділити деякі спільні та відмінні характеристики. Всі три платформи мають візуальні редактори, які дозволяють користувачам легко створювати вебсайти без знань програмування. Водночас, вони мають різні цільові аудиторії та функціональні можливості.

Tilda та Wix, зокрема, підходять для непрофесіоналів та малого бізнесу, які шукають простий спосіб створити ефективний сайт без великих витрат часу та коштів. Ці платформи забезпечують простоту використання, широкий вибір готових шаблонів та адаптивний дизайн.

Webflow, з іншого боку, пропонує більше можливостей для професійних вебдизайнерів та розробників. Його функціональність і гнучкість дозволяють створювати більш складні вебпроєкти з унікальним дизайном та функціоналом.

Щодо підтримки бізнесу, платформи Tilda, Wix та Webflow допомагають компаніям створювати власні вебсайти для просування своїх товарів та послуг. Конструктори дозволяють швидко створити сайти з різними функціями, такими як форми зворотного зв’язку, онлайн-магазини, блоги тощо. Це важливо для успіху сучасного бізнесу, оскільки вебприсутність може покращити доступність та видимість компанії в інтернеті.

Крім того, ці платформи можуть служити навчальним інструментом для тих, хто хоче дізнатися про веброзробку та дизайн. Наприклад, Webflow пропонує навчальні матеріали та ресурси, які допомагають новачкам та професіоналам розвивати свої навички у веброзробці та дизайні.

Отже, аналізуючи схожі платформи, можна зробити висновок, що вони надають широкий спектр можливостей для різних користувачів та цілей. При виборі платформи важливо враховувати специфіку проєкту, цільову аудиторію, бюджет та особисті навички. Таким чином, ви зможете знайти найбільш підходящу платформу для вашого вебпроєкту, забезпечити його успішне функціонування та розвиток.

1.3 Постановка задачі

Таким чином, розробка платформи для створення вебсторінок є актуальним завданням. Ця платформа може спростити процес створення сайту, зробити його доступним не тільки для професіоналів, але й для тих, хто вперше зустрічається з цим завданням. Крім того, платформа може допомогти оптимізувати роботу сайтів, що сприятиме збільшенню продуктивності та ефективності.

Об’єктом роботи є моделювання та розробка платформи для створення вебсторінок.

Метою роботи є розробка мінімального життєздатного продукту (MVP) платформи для створення вебсторінок. За допомогою технології необхідно реалізувати зручну платформу, де можна буде легко та інтуїтивно створювати вебсторінки різного направлення.

Для досягнення мети необхідно вирішити такі завдання:

– розробити систему авторизації для користувачів платформи, що дозволить відслідковувати їхній прогрес та забезпечити безпечний доступ до їх проєктів;

– розробити набір готових блоків, які можуть бути легко інтегровані та комбіновані у вебсторінку за допомогою «drag and drop» інтерфейсу;

– створити можливість для користувачів зберігати та перевикористовувати створені блоки, що полегшує процес створення нових вебсторінок та забезпечує зручність роботи;

– розробити систему публікації готових вебсторінок в інтернет на піддомені, що дозволить користувачам легко розгортати свої проєкти та надавати доступ до них іншим користувачам.

В результаті реалізації зазначених цілей, платформа, що буде розроблена в рамках цієї роботи, спрямована на спрощення та оптимізацію процесу створення вебсторінок для користувачів з різним рівнем технічної підготовки. Завдяки зручному та інтуїтивно зрозумілому інтерфейсу, користувачі зможуть

створювати власні вебсайти без необхідності залучення професійних розробників та дизайнерів.

Крім того, платформа буде корисною для освітніх цілей, допомагаючи новачкам в галузі веброзробки отримати перше уявлення про процес створення сайтів та основні принципи їх функціонування.

Останнім, але не менш важливим аспектом є вплив платформи на розвиток бізнесу. Завдяки швидкому та ефективному створенню вебсторінок, підприємства зможуть зосередитися на своїх основних функціях та досягати кращих результатів у продажах, просуванні та залученні нових клієнтів.

Таким чином, розробка та моделювання платформи для створення вебсторінок, заснованої на принципах «drag and drop» та реалізації набору готових блоків, може стати значущим внеском у сферу веброзробки та бізнесу, сприяючи легкому створенню вебсайтів та доступу до онлайн-присутності для широкого кола користувачів.

2 МОДЕЛЮВАННЯ АРХІТЕКТУРИ ПЛАТФОРМИ ДЛЯ СТВОРЕННЯ ВЕБСТОРІНОК

2.1 Огляд існуючих архітектурних рішень

Архітектура має величезне значення у процесі створення програмного забезпечення, оскільки вона визначає швидкість розробки, здатність до підтримки, тестування, адаптивність та надійність програмного продукту. В сучасному світі веброзробки існує концепція «Чистої архітектури» [9], яка представляє собою комплекс порад і настанов для створення програм, що включає найкращі практики, такі як шаблони проєктування та принципи SOLID. Використання цієї парадигми дозволяє розробляти чистий, стабільний та легко тестований програмний код, який можна без проблем розширювати та ефективно підтримувати.

Чиста архітектура є сучасним стилем архітектури програмного забезпечення, який був розроблений відомим експертом Робертом Мартіном (рис. 2.1). За останні десятиліття його ім’я стало відомим серед розробників завдяки численним виданням, таким як книга «Чистий код» та принципам SOLID, які активно використовуються та обговорюються програмістами у всьому світі. Мартін також широко відомий своїми лекціями та публікаціями з області об’єктно-орієнтованого програмування.

Важливо зазначити, що запропонована ним структура архітектури не орієнтована на об’єктно-орієнтовану парадигму, хоча може бути реалізована за допомогою об’єктно-орієнтованої мови програмування, використовуючи об’єктно-подібні конструкції і синтаксис, основна частина пропозиції базується на процедурному мисленні.

Цей підхід до архітектури програмного забезпечення дозволяє створювати більш чистий, зрозумілий і легко підтримуваний код, що в результаті веде до створення більш стабільних та надійних програмних продуктів. Тому вивчення і впровадження чистої архітектури є важливим

кроком для професійного розвитку програміста і покращення якості програмного забезпечення в цілому.

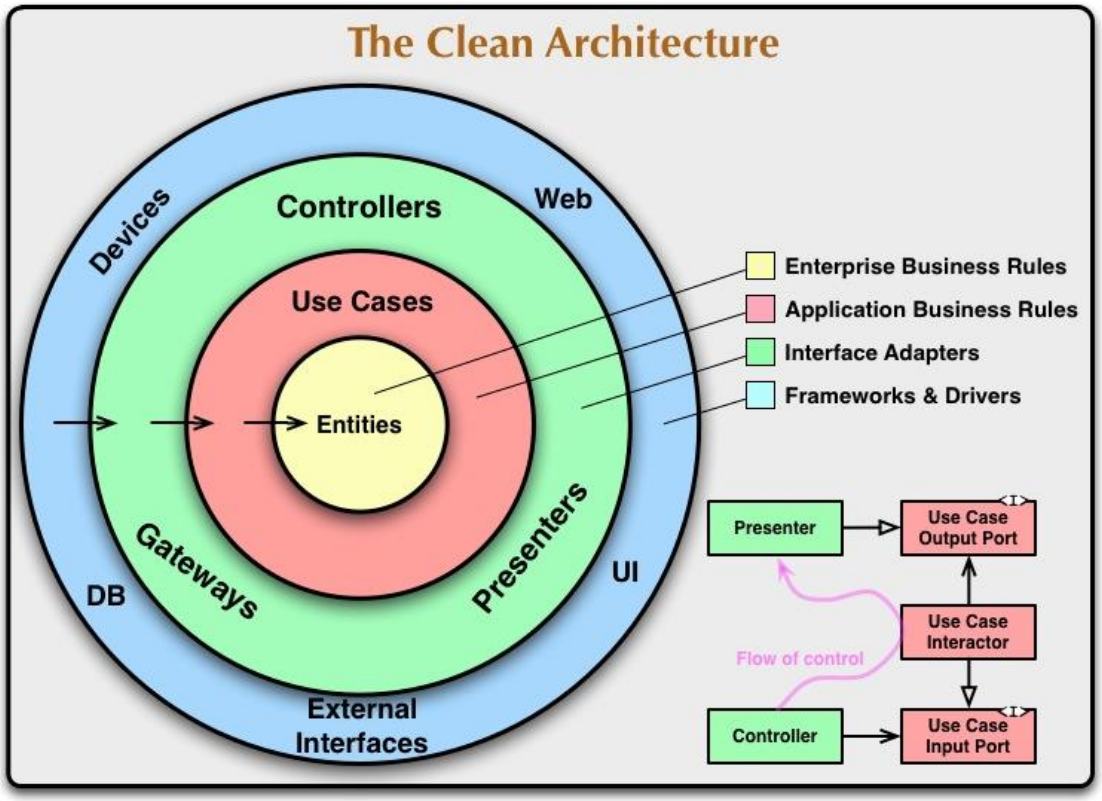

Рисунок 2.1 – Приклад чистої архітектури

Розпочнемо з переліку декількох пов’язаних архітектурних шаблонів, з яких натхненний Мартін і створив ідею чистиї архітектури, а саме:

– «Гексагональна архітектура» Алістера Кокберна [10];
– «Архітектура цибулі» Джеффрі Палермо [11].

Гексагональна архітектура, також відома як порти та адаптери, була представлена Алістером Кокберном. Ця архітектура спрямована на створення програмного забезпечення, яке відрізняється високим рівнем гнучкості та можливостями для тестування. Гексагональна архітектура використовує шарувату структуру, де бізнес-логіка програми ізольована від зовнішніх факторів, як-от взаємодія з користувачем, базами даних, вебсерверами тощо (рис. 2.2).

Рисунок 2.2 – Приклад гексагональної архітектури

Принцип гексагональної архітектури полягає в розділенні бізнес-логіки (або ядра застосунка) від зовнішніх факторів, які можуть впливати на неї. Це досягається за допомогою створення абстракцій, які ізолюють ядро застосунка від інфраструктури, яка обслуговує вхідні та вихідні запити (наприклад, взаємодії з базою даних, мережевими запитами, інтерфейсом користувача і т.д.). Ці абстракції називаються «портами», а реалізації, які обробляють конкретні деталі інфраструктури, називаються «адаптерами».

Головна мета гексагональної архітектури – забезпечити гнучкість, дозволяючи зміни зовнішніх інтерфейсів без впливу на бізнес-логіку. Це особливо корисно в динамічному середовищі розробки, де технології та вимоги можуть швидко змінюватися. За допомогою гексагональної архітектури розробники можуть легко замінювати або модифікувати зовнішні інтерфейси, не змінюючи бізнес-логіку. Наприклад, якщо компанія вирішує змінити базу даних або впровадити новий вебсервіс, вона може це зробити без ризику вплинути на внутрішню логіку застосунка.

Важливим аспектом гексагональної архітектури є покращення тестового процесу. Оскільки бізнес-логіка відокремлена від зовнішніх факторів, вона може бути протестована незалежно і в ізоляції. Це означає, що можна використовувати тестові адаптери для симуляції зовнішнього середовища, що дозволяє проводити ефективні юніт-тести без необхідності налаштовувати додаткове тестове середовище.

Гексагональна архітектура вирішує декілька ключових проблем, з якими зіштовхуються розробники:

– зміна технологій: у сучасному світі технології швидко змінюються. Гексагональна архітектура дозволяє розробникам адаптуватися до цих змін без впливу на бізнес-логіку;

– тестування: завдяки відокремленню бізнес-логіки від інфраструктури, гексагональна архітектура полегшує створення надійних тестів;

– підтримка та розширюваність: гексагональна архітектура забезпечує чітку організацію коду, що сприяє легшому розумінню, підтримці та розширенню системи.

В той же час, гексагональна архітектура може бути відносно складною для розуміння та імплементації, особливо для команд з меншим досвідом. Це також може призвести до певної надмірної складності для простих проєктів, де можуть вистачити менш складні архітектурні підходи.

Додатково, важливим є розуміння того, що гексагональна архітектура – це не конкретна реалізація, а підхід до дизайну системи. Це означає, що специфіка реалізації може суттєво відрізнятися в залежності від конкретних вимог та контексту застосунка. Існує безліч способів імплементувати гексагональну архітектуру, і це може вимагати значної творчості та технічної проникливості від команди розробників.

На відміну від традиційних монолітичних архітектур, гексагональна архітектура прагне до централізації бізнес-логіки та ізолювання її від зовнішніх залежностей. Це відкриває можливості для більшої гнучкості та

повторного використання коду, покращуючи при цьому стійкість програмного забезпечення до змін.

Архітектура цибулі, розроблена Джеффрі Палермо, є альтернативним підходом до організації коду в програмному забезпеченні. Цей підхід використовує концепцію «цибулі», яка представляє собою шари абстракції, що накладаються один на одного. В центрі цибулі знаходиться бізнес-логіка (домен), а інші шари розміщені навколо неї (рис. 2.3). Основна мета архітектури цибулі полягає в забезпеченні чіткої організації коду, що забезпечує високий рівень зв'язності та низький рівень залежності між компонентами.

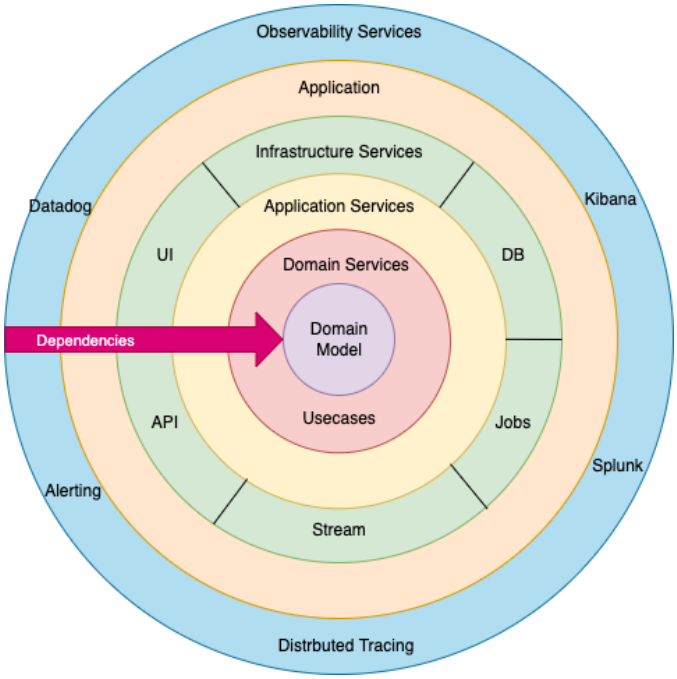

Рисунок 2.3 – Приклад архітектури цибулі

Принцип архітектури цибулі базується на ідеї розділення коду на концентричні шари, кожен з яких має свої обов'язки та залежності. Найважливіший шар – це домен (бізнес-логіка), який не має зовнішніх залежностей і є незалежним від інфраструктури. Навколо домену розташовані шари абстракцій, які слугують для обробки взаємодії з інфраструктурними сервісами, такими як бази даних, вебсервери та інші зовнішні ресурси.

Співвідношення залежностей у архітектурі цибулі спрямоване від зовнішніх шарів до внутрішніх. Це означає, що внутрішні шари не мають прямих залежностей від зовнішніх шарів, але зовнішні шари можуть залежати від внутрішніх. Таким чином, код, який відповідає за бізнес-логіку, залишається чистим та нескаладним, і не містить ніяких прямих залежностей від інфраструктури та інших зовнішніх ресурсів. Це дозволяє зосередитися на бізнес-логіці, поліпшуючи читабельність та підтримку коду, а також спрощуючи тестування.

Архітектура цибулі допомагає вирішити декілька проблем, з якими зіштовхуються розробники програмного забезпечення:

– чітка організація коду: розділення коду на шари забезпечує високий рівень зв’язності та низький рівень залежності між компонентами. Це полегшує розуміння, підтримку та розширення програмного забезпечення;

– спрощене тестування: оскільки бізнес-логіка ізольована від зовнішніх шарів та залежностей, тестування стає простішим, швидшим та більш надійним. Юніт-тести можна розробляти безпосередньо для доменного шару, без необхідності створення зовнішніх залежностей;

– гнучкість: архітектура цибулі полегшує зміни в програмному забезпеченні, оскільки зміни в зовнішніх шарах не впливають на бізнес-логіку. Це дозволяє розробникам швидше адаптуватися до нових вимог та технологій.

Загалом, архітектура цибулі пропонує відмінний підхід до проєктування та створення програмного забезпечення, який забезпечує гнучкість, легкість тестування та можливість адаптації до змін. Використання цього підходу може допомогти командам створювати більш стабільні, надійні та підтримувані системи, що у свою чергу може підвищити ефективність розробки та забезпечити більшу цінність для користувачів [12-18].

Для реалізації поставленої задачі було вирішено використовувати триланкову архітектуру, яка складається з таких компонентів: клієнт, сервер і база даних. Схема даної архітектури зображена на рисунку 2.4.

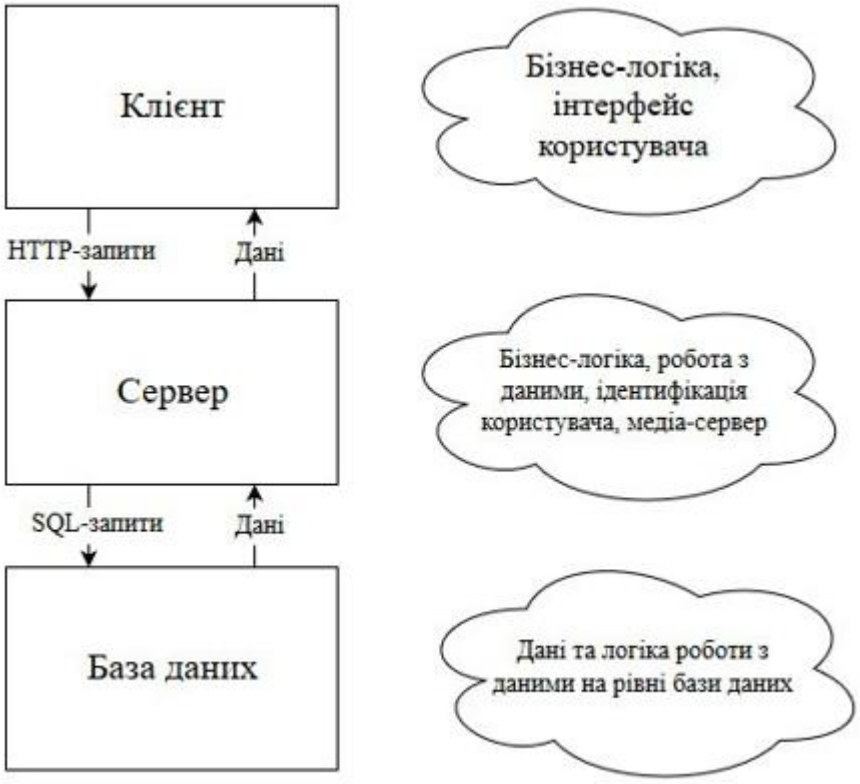

Рисунок 2.4 – Триланкова архітектура програмного застосунку

Триланкова архітектура (також відома як трьохрівнева архітектура) є однією з найпоширеніших підходів до проєктування розподілених систем. Вона включає три основні компоненти: клієнтську частину (або користувацький інтерфейс), серверну частину (або бізнес-логіку) та базу даних (або сховище даних). Ця архітектура допомагає вирішувати ряд проблем, пов’язаних з масштабуванням, підтримкою, безпекою та ефективністю систем [19].

Принцип триланкової архітектури полягає в розділенні відповідальності між різними рівнями системи. Кожен з трьох рівнів має свою специфічну функцію, що дозволяє легко розширювати, модифікувати та підтримувати систему.

Клієнтська частина (презентаційний рівень): Цей рівень відповідає за представлення даних користувачеві та збір відгуків від користувача. Він може включати вебсторінки, мобільні додатки або інші користувацькі інтерфейси.

Завдання клієнтської частини – надати зручний та інтуїтивний доступ до функціональності системи.

Серверна частина (рівень бізнес-логіки): Цей рівень відповідає за обробку даних, отриманих від клієнтів, та взаємодію з базою даних. Він містить бізнес-логіку, правила та алгоритми, які керують обробкою даних. Серверна частина може включати різні служби, компоненти та модулі, які працюють разом для забезпечення правильного функціонування системи. Завдання серверної частини – ефективно обробляти запити від клієнтів та забезпечувати безпечне зберігання та доступ до даних.

База даних (рівень даних): Цей рівень відповідає за зберігання, відновлення та управління даними, які використовуються в системі. Він може включати реляційні або нереляційні бази даних, файлові сховища або інші форми зберігання даних. Завдання бази даних – надавати швидкий та надійний доступ до даних для серверної та клієнтської частин.

Переваги триланкової архітектури:

– масштабування: розділення відповідальності між різними рівнями дозволяє незалежно масштабувати кожен з них, забезпечуючи гнучкість та відповідність вимогам системи;

– підтримка та розвиток: триланкова архітектура полегшує підтримку та розвиток системи, оскільки зміни можуть вноситися на відповідних рівнях без значного впливу на інші компоненти;

– безпека: розділення бізнес-логіки та даних може поліпшити безпеку системи, оскільки доступ до даних може контролюватися та обмежуватися на рівні сервера.

В цілому, триланкова архітектура є добре зарекомендувала себе підхід до проєктування розподілених систем, пропонуючи гнучкість, масштабування та розділення відповідальності. Вона є відмінним вибором для розробки платформи для створення вебсторінок.

2.2 Особливості бази даних платформи для створення вебсторінок

У сучасному світі дані стали невід’ємною частиною нашого життя, зокрема в наукових, бізнес та технічних сферах. Значення та використання даних продовжують зростати, і розуміння того, як вони зберігаються та обробляються, стає все більш важливим [20].

Дані можуть існувати у різних формах, таких як графіки, звіти, таблиці, текст і багато інших, представляючи різноманітні види інформації, які можна легко отримувати, оновлювати, аналізувати та представляти через систематизоване або структуроване сховище індексованої інформації.

Великі сховища даних відомі як бази даних, які можуть бути прикладом публічної бібліотеки, що зберігає книги. Бази даних – це комп’ютерні системи, які зберігають, організовують, захищають та надають доступ до даних. Системи, які контролюють бази даних, відомі як системи управління базами даних

Всередині бази даних, дані зберігаються у таблицях, які складаються з рядків та стовпців, та індексуються таким чином, що забезпечує простий пошук відповідної інформації. З появою нової інформації, дані оновлюються, розширюються та видаляються. Різні процеси баз даних самі створюють і оновлюють записи, виконуючи запити.

База даних – це сукупність відомостей, які зберігаються, обробляються та надаються користувачам у структурованому вигляді. Бази даних можуть бути класифіковані за різними критеріями, такими як їхній структурі, способу зберігання та обробки даних. Основні види баз даних включають:

Реляційні бази даних – це системи зберігання даних, які організовують інформацію у структурованому вигляді за допомогою таблиць, стовпців та рядків. Вони ґрунтуються на реляційній моделі, розробленій Едгаром Коддом у 1970-х роках, яка передбачає використання реляцій (таблиць) та атрибутів (стовпців) для представлення даних та їх взаємозв’язків. Реляційні бази даних

використовують мову SQL (Structured Query Language) для створення, оновлення, видалення та отримання даних.

Найпопулярнішими реляційними системами управління базами даних (СУБД) є MySQL, PostgreSQL, Microsoft SQL Server та Oracle Database. Вони надають широкий спектр функціональності для роботи з даними, такі як транзакції, індексування, обмеження на цілісність даних та можливість використання зовнішніх процедур та тригерів для автоматичної обробки даних та підтримки бізнес-логіки.

Нереляційні бази даних, також відомі як NoSQL (Not only SQL) бази даних, – це системи зберігання даних, які не використовують традиційну реляційну модель з таблицями, стовпцями та рядами. Замість цього, NoSQL бази даних пропонують більш гнучкі та різноманітні структури даних, такі як ключ-значення, документи, колонки або графи. Це дозволяє вони легше масштабуватися, забезпечує високу продуктивність та зручність роботи з непостійними або неструктурованими даними.

Один із прикладів використання нереляційної бази даних – додаток для обробки та аналізу великих обсягів соціальних медіа даних. Такі дані можуть бути різноманітними, неструктурованими та змінюватися з часом. Використання бази даних на основі документів, такої як MongoDB, дозволяє зберігати дані у форматі JSON, що спрощує роботу з різними типами даних та динамічної структури. Крім того, такі бази даних можуть легко масштабуватися горизонтально для підтримки зростання обсягів даних та кількості запитів.

Деякі найвідоміші приклади NoSQL баз даних включають MongoDB (документорієнтована), Cassandra (колоночна), Redis (ключ-значення) та Neo4j (графова). Вони надають високу продуктивність та масштабованість, а також спеціалізовані можливості для роботи з різними типами даних та сценаріїв застосування. Втім, вони можуть мати менш виразні можливості щодо обмежень цілісності, транзакцій або складних запитів порівняно з реляційними базами даних.

Графоорієнтовані бази даних – це спеціалізовані системи зберігання даних, які використовують графову модель для представлення та зберігання даних. Вони зосереджуються на відносинах між елементами даних та надають засоби для ефективного виконання складних запитів, що вимагають глибокого аналізу зв’язків. Графоорієнтовані бази даних складаються з вузлів (вершини графа) та ребер (зв’язки між вузлами). Вузли можуть представляти різні типи сутностей, а ребра – різні типи відносин між ними.

Один із прикладів використання графоорієнтованої бази даних – система рекомендацій, яка аналізує взаємозв’язки між користувачами, продуктами та їх взаємодією, такими як відгуки та оцінки. Графоорієнтована база даних, така як Neo4j, дозволяє ефективно знаходити шляхи між сутностями, виявляти закономірності та виділяти групи схожих елементів. Це допомагає розробити високоякісні рекомендації, враховуючи різні аспекти взаємодії користувачів та продуктів. Графоорієнтовані бази даних також можуть бути використані для аналізу соціальних мереж, наукових даних, фінансових операцій та інших доменів, де відносини між сутностями є ключовим аспектом даних.

Персональна база даних використовується для зберігання даних, що зберігаються на персональних комп’ютерах, менших за розміром і ними легко керувати. Дані в основному використовуються одним і тим же відділом компанії, і доступ до них має невелика група людей.

Мультимодальна база даних – це тип платформи обробки даних, що підтримує безліч моделей даних, що визначають, як слід організовувати та впорядковувати певні знання та інформацію в базі даних

Використання PostgreSQL для створення вебсторінок є правильним варіантом, оскільки ця реляційна база даних надає відмінні можливості для зберігання та обробки структурованих даних, а також підтримує тип даних jsonb. Завдяки jsonb, PostgreSQL дозволяє працювати з непостійними та неструктурованими даними, які можуть бути важливими для вебсторінок зі складною структурою та динамічним контентом.

Таким чином, PostgreSQL поєднує найкращі сторони реляційних баз даних, такі як надійність, безпека та виразність, з можливостями для роботи з даними, подібними до тих, які надають нереляційні бази даних. Завдяки цьому, PostgreSQL стає ідеальним вибором для платформи для створення вебсторінок, які вимагають як структурованого, так і неструктурованого зберігання даних, та забезпечує гнучкість та продуктивність, необхідні для успішної реалізації проєкту [21].

Під час розробки платформи було спроєктовано базу даних (рис. 2.5).

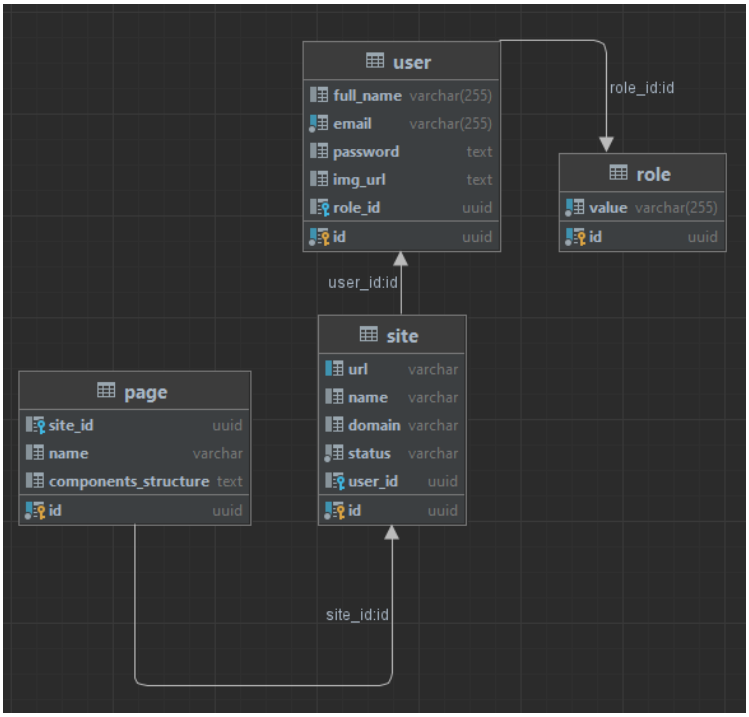

Рисунок 2.5 – Схема бази даних платформи для створення вебсторінок

Як видно з бази даних, платформі необхідно чотири таблиці, які зберігають інформацію користувачів і сервісу.

Головною таблицею системи є site, де зберігається головна інформація про створенні сайти користувачів. Не менш значущою таблицею є page яка зберігає основну структуру сторінки сайту. Інші таблиці містять інформацію при користувачів системи.

Таблиці мають зв’язки типу один-до-одного та один-до-багатьох. Зв’язок «один-до-одного» передбачає, що одній строчці з першої таблиці

відповідає лише одна строчка з іншої, та навпаки. Наприклад – одній строчці з таблиці user відповідає лише одна строчка з таблиці role, бо один користувач може мати лише одну роль.

Зв’язок «один-до-багатьох» передбачає, що одній строчці з першої таблиці може відповідати безліч строчок з іншої, але навпаки лише одній строчці з другої таблиці відповідає лише одна строчка з першої. Наприклад – одній строчці з таблиці site відповідає безліч строчок з таблиці page, бо один сайт може мати безліч сторінок, тоді як одній строчці з таблиці page відповідає лише одна строчка з таблиці site, бо одна сторінка може мати відношення лише до одного сайту.

Для розробки системи було розроблено чотири основних таблиць. Розглянемо усі таблиці детально.

Першою є таблиця role (табл. 2.1).

Таблиця 2.1 – Таблиця role БД платформи для створення вебсторінок

Data	**Type**	**Autocreate**	**Example**
id	uuid	Yes	d75450ec-38e6-4eab-98bb-ceafb5d4f134
value	varchar(255)	No	USER

Дана таблиця містить дані про значення ролі та ідентифікатор цієї ролі.

В таблиці 2.2 приведено таблицю user.

Таблиця 2.2 – Таблиця user БД платформи для створення вебсторінок

Data	**Type**	**Autocreate**	**Example**
id	uuid	Yes	d75450ec-38e6-4eab-98bb-ceafb5d4f134
full_name	varchar(255)	No	Steve John
email	varchar(255)	No	vladyslav@email.co

Продовження таблиці 2.2

Data	**Type**	**Autocreate**	**Example**
password	text	No	$2a$05$Grkkc7Gri9oJXfRH4bjj/O2shmBBTTpxSHTAtchwYSU2qk/YtKMvG
img_url	text	No	https://domain/image.png
role_id	uuid	No	96bf35ac-effc-402d-a1b6-284b39c90fc7

Дана таблиця містить дані про користувача платформи, його повне ім’я, email, захешований пароль, посилання на зображення, яке відображається на сторінці профілю, унікальний ідентифікатор користувача та ідентифікатор ролі користувача.

В таблиці 2.3 приведено таблицю site.

Таблиця 2.3 – Таблиця site БД платформи для створення вебсторінок

Data	**Type**	**Autocreate**	**Example**
id	uuid	Yes	d75450ec-38e6-4eab-98bb-ceafb5d4f134
url	varchar(255)	No	steve
name	varchar(255)	No	steve project
domain	varchar(255)	No	PLIA
status	varchar(255)	Yes	UNPUBLISHED
user_id	uuid	No	96bf35ac-effc-402d-a1b6-284b39c90fc7

Дана таблиця містить дані про створені сайти користувача, посилання сайту, ім’я проєкту, тип домену проєкта це може бути як піддомен платформи

так і унікальний домен користувача, статус публікації сайту, унікальний ідентифікатор сайту та ідентифікатор користувача якому приналежить цей сайт.

В таблиці 2.4 приведено таблицю page.

Таблиця 2.4 – Таблиця page БД платформи для створення вебсторінок

Data	**Type**	**Autocreate**	**Example**
id	uuid	Yes	d75450ec-38e6-4eab-98bb-ceafb5d4f134
name	varchar(255)	No	Home
components_structure	jsonb	Yes	{"id":"body","component":"Body","children":[]}
site_id	uuid	No	96bf35ac-effc-402d-a1b6-284b39c90fc7

Дана таблиця містить дані про створені сторінки користувача, ім’я сторінки, компонентну структуру даних за якою генерується сторінка, унікальний ідентифікатор сторінки та ідентифікатор сайту.

2.3 Проєктування модулів та взаємодії компонентів системи

У сучасному світі розробки програмного забезпечення, проєктування модулів та взаємодії компонентів системи відіграє надзвичайно важливу роль. Ефективні архітектурні методології та використання монорепозиторіїв можуть сприяти кращому управлінню кодом, поліпшенню продуктивності розробників та створенню масштабованих та стабільних програмних продуктів. Розуміння того, як правильно проєктувати та взаємодіяти з компонентами, вирішує купу проблем, з якими можуть зіткнутися команди під час процесу розробки.

Монорепозиторії є централізованими сховищами для управління кодом, залежностями та конфігурацією. Він може включати декілька пакетів, модулів, додатків, бібліотек та служб, які працюють разом. Вони сприяють спільному використанню ресурсів між різними проєктами, спрощують процеси розгортання та автоматизації та дозволяють командам працювати над різними частинами системи одночасно, не втрачаючи контролю над стабільністю та якістю коду. Застосування монорепозиторіїв у проєктуванні модулів та взаємодії компонентів є ключовим для підтримки стабільної та ефективної архітектури [22].

Монорепозиторії вирішують ряд проблем, зокрема це спрощення управління залежностями, оскільки всі пакети та модулі зберігаються в одному репозиторії, це полегшує управління залежностями та оновлення версій. З розробниками, що працюють над різними частинами проєкту, монорепозиторій спрощує відстеження змін та координацію роботи. Монорепозиторії полегшують розробку спільних бібліотек, компонентів та служб, що можуть бути використані в різних частинах проєкту.

У контексті проєктування вебзастосунків монорепозиторії можуть допомогти організувати код та ресурси для багаторівневих архітектур, таких як клієнт-сервер, де код для вебклієнта та сервера зберігається в одному репозиторії. Це полегшує розробку, налаштування та підтримку проєкту, забезпечуючи єдине місце для управління залежностями, конфігурацією та іншими ресурсами.

На даний момент існує два популярних інструмента для роботи з монорепозиторіями – це Nx та Lerna.

Nx – це інструмент, розроблений компанією Nrwl, який допомагає розробникам створювати масштабовані та ефективні монорепозиторії. Він пропонує ряд функцій та можливостей для роботи з різними фреймворками, такими як Angular, React, Solid.js та Node.js.

Ключовими особливостями Nx є єдине місце для конфігурації: Nx надає єдине місце для управління конфігурацією проєкту, що полегшує роботу з

багатьма проєктами та пакетами. Забезпечення кешування та інкрементної збірки. Nx оптимізує процес збірки та тестування, використовуючи кешування та інкрементну збірку, що забезпечує швидше виконання завдань. Вбудовані схеми для популярних фреймворків: Nx містить вбудовані схеми для створення додатків, бібліотек та служб, що працюють з Angular, React, Solid.js, Node.js та іншими фреймворками.

Lerna – це інструмент для управління монорепозиторіями JavaScript, який допомагає розробникам керувати залежностями та публікацією пакетів. Основні характеристики Lerna включають, управління залежностями Lerna спрощує управління залежностями між пакетами в монорепозиторії, автоматизуючи оновлення версій та встановлення потрібних пакетів. Lerna допомагає автоматизувати процес публікації пакетів у репозиторіїх, таких як npm, забезпечуючи послідовне та контрольоване випуск оновлень. Також Lerna надає можливість виконання скриптів та команд на рівні монорепозиторію, спрощуючи автоматизацію різних процесів розробки.

Враховуючи різні аспекти, що були описані раніше, можна зробити висновок, що використання Nx є більш правильним варіантом для управління монорепозиторіями, порівняно з Lerna. Nx пропонує більше можливостей для оптимізації розробки та підтримки вебзастосунків, оскільки він містить вбудовані схеми для популярних фреймворків, інкрементну збірку та кешування.

Окрім того, Nx має більше ресурсів для навчання та документації, що може допомогти розробникам швидко освоїти інструмент та створювати рішення з високою продуктивністю та ефективністю. Це робить Nx більш привабливим для розробки програмного забезпечення з різними фреймворками та технологіями.

Під час розробки платформи була спроєктована взаємодія основних модулів платформи для створення вебсторінок (рис. 2.6).

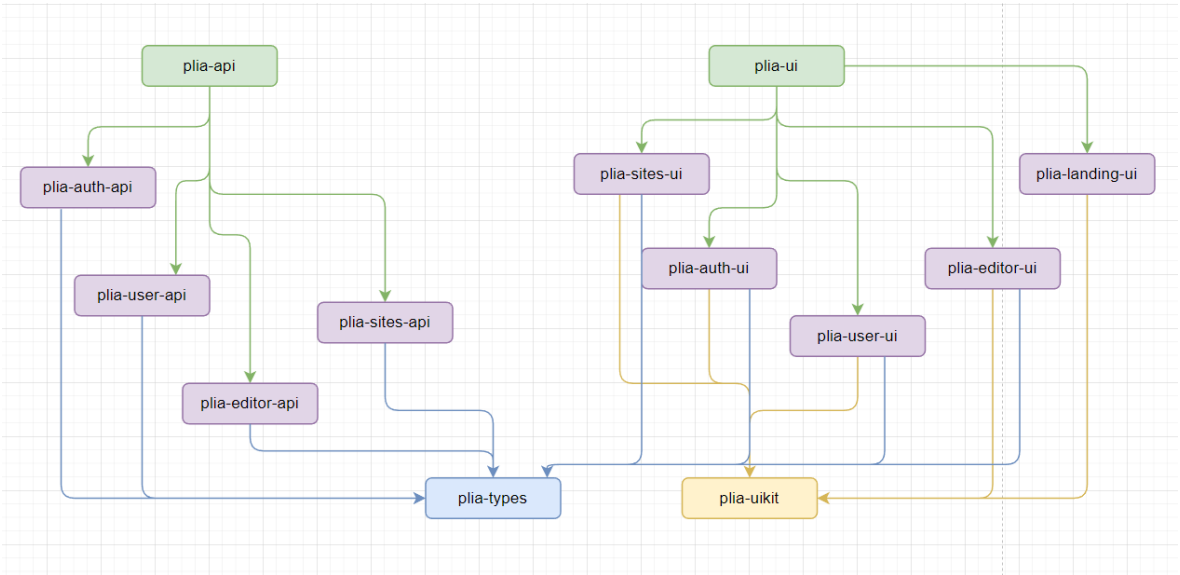

Рисунок 2.6 – Граф взаємодії основних модулів платформи для створення вебсторінок

Як можна побачити з графу проєкту, платформі потрібно два основних застосунку – це plia-api та plia-ui. Завдяки NX та концепції монорепозиторія, ми маємо змогу зберігати код як серверної так і клієнтської частини в одному проєкті.

Plia-api це серверна частина платформи яка використовує чотири модуля, в яких знаходиться основна серверна логіка застосунку. Розглянемо кожен модуль більш детально.

Plia-auth-api – це серверний модуль який відповідальний за авторизацію та реєстрацію користувачів на сервері.

Plia-user-api – це серверний модуль який відповідальний за додавання, видалення, редагування та перегляд усієї інформації про користувачів платформи.

Plia-editor-api – це серверний модуль який відповідальний за додавання, видалення, редагування та перегляд усіх створених сторінок користувача платформи. Також його задачею є створення HTML файлів вебсторінок.

Plia-sites-api – це серверний модуль який відповідальний за додавання, видалення, редагування та перегляд усіх сайтів користувача платформи.

Plia-ui – це клієнтська частина платформи яка використовує п’ять модулів в яких знаходиться основна клієнтська логіка застосунку. Розглянемо кожен модуль більш детально.

Plia-landing-ui – це клієнтський модуль в якому знаходиться головна сторінка застосунку яка описує що це за платформа та які можливості вона надає. З цього модулю користувач має змогу увійти до платформи або зареєструватися.

Plia-auth-ui – це клієнтський модуль в якому знаходяться сторінки реєстрації та входу в вебзастосунок. Він відповідальний за відображення помилок при реєстрації або вході, а при успішному вході або реєстрації перенаправлення на сторінку створення або редагування існуючих сайтів користувача платформи.

Plia-sites-ui – це клієнтський модуль в якому знаходиться сторінка створення або перегляду існуючих сайтів користувача. У цьому модулі користувач може змінити ім’я сайту або його адрес в інтернеті. Також з цього модуля користувач може потрапити до сторінки профілю або до серця платформи, а саме візуального редактора сторінок.

Plia-editor-ui – це клієнтський модуль в якому знаходиться візуальний редактор сторінок. У ньому користувач може легко додавати нові блоки за допомогою «drag and drop» або міняти їх місцями. Також користувач має змогу змінювати зовнішній вигляд блоків з яких і створюється вебсторінка.

Plia-user-ui – це клієнтський модуль в якому знаходиться сторінка профілю користувача. На якій він може переглянути свою інформацію на платформі, змінити пароль або email.

Також проєкт має два модуля спільного користування – це plia-types та plia-ui. Розглянемо їх більш детально

Plia-types – це модуль спільного користування в якому розташовані усі інтерфейси або типи які можуть бути використані як і на серверної частині проєкту так і на клієнтської. Це дуже зручно, тому що розробнику не потрібно копіювати типи з серверної частини на клієнтську або навпаки.

Plia-uikit – це модуль спільного користування в якому розташовані усі UI компоненти проєкту, які використовуються в клієнтських модулях. Його основна мета це надання «дурних» компонентів які нічого не повинні знати про специфіку застосунку, а повинні тільки відображати дані які в них надходять. Перевага такого підходу полягає в тому що цей модуль може бути легко перевикористан в інших проєктах або модифікований більш досвідченими розробниками.

Також немало важливим аспектом при створенні великих та розширюваних вебзастосунків є використання методологій структурування коду. Однією з таких є FSD (Feature Slice Design) він зосереджується на розбитті проєкту на незалежні, легко зрозумілі функціональні частини, відомі як «фічі». Також він є відповіддю на проблеми, пов'язані з традиційними підходами до організації коду, такими як групування за типом (компоненти, сервіси, модулі тощо). Головна мета FSD методології полягає у полегшенні розробки, розширення, тестування й підтримки коду.

У FSD проєкт складається з шарів, кожен шар складається зі зрізів, а кожен зріз – з сегментів (рис. 2.7) [23].

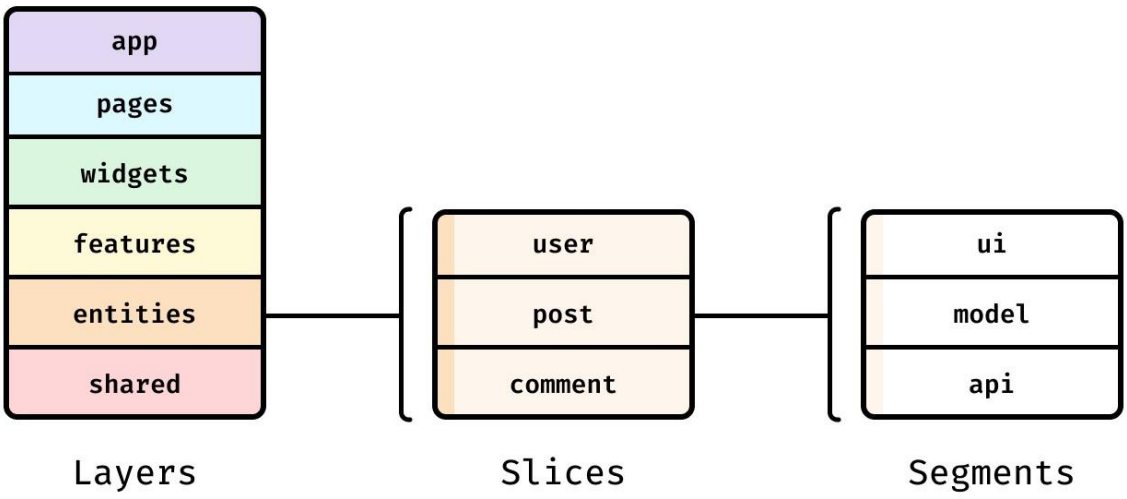

Рисунок 2.7 – Основні частини FSD

Шари стандартизовані для всіх проєктів і розташовані вертикально. Модулі на одному шарі можуть взаємодіяти лише з модулями з шарів, розташованих строго нижче. Наразі їх шість.

Shared – функціонал для багаторазового використання, відокремлений від специфіки проєкту/бізнесу (наприклад, UIKit, бібліотеки, API).

Entities – бізнес-сутності (наприклад, Користувач, Продукт, Замовлення).

Features – взаємодія з користувачем, дії, які приносять бізнес-цінність користувачеві (наприклад, SendComment, AddToCart, UsersSearch).

Widgets – композиційний рівень для об’єднання сутностей та функцій у змістовні блоки (наприклад, IssuesList, UserProfile).

Pages – композиційний рівень для побудови повних сторінок з сутностей, функцій та віджетів.

App – налаштування, стилі та провайдери для всього застосунку.

Потім є зрізи, які розділяють код за бізнес-доменами. Це полегшує навігацію у кодовій базі, оскільки логічно пов’язані модулі розташовуються близько один до одного. Зрізи не можуть використовувати інші зрізи на тому ж рівні, і це допомагає досягти високої згуртованості та низького рівня зв’язку.

Кожен слайс, у свою чергу, складається з сегментів. Це крихітні модулі, які призначені для розділення коду всередині зрізу за його технічним призначенням. Найпоширеніші сегменти – це ui, model (сховище, дії), api та lib (утиліти), але також можна опустити деякі з них або додати більше.

Застосування методології Feature Slice Design дозволяє створювати гнучкі, масштабовані та легко підтримувані вебзастосунки. Цей підхід пропонує ефективну структуру коду, яка полегшує розробку та координацію роботи команди. Організація коду на основі функціональності та відповідальності сприяє створенню чистого, зрозумілого та стабільного програмного продукту.

3 РОЗРОБКА ПЛАТФОРМИ ДЛЯ СТВОРЕННЯ ВЕБСТОРІНОК

3.1 Обґрунтування вибору середовища програмної реалізації

Для реалізації програмного застосунку було використано мову програмування JavaScript. JavaScript є однією з основних технологій Інтернету, поряд з HTML і CSS, які роблять вебсторінки інтерактивними. Ця мова програмування створена у 1995 році Бренданом Айхом під час його роботи в Netscape Communications, з метою додавання «живості» статичним вебсторінкам [24].

Історично, до появи JavaScript, вебсторінки були в основному статичні, здатні тільки відображати інформацію, яку сервер надсилає браузеру. Взаємодія з користувачем обмежувалася заповненням форм і навігацією по посиланням. Всі динамічні процеси відбувались на сервері, наприклад, за допомогою мови програмування PHP. PHP був використаний для створення динамічних вебсторінок з 1994 року, але обмежувався серверними процесами і не міг впливати на взаємодію на клієнтській стороні.

JavaScript змінив це, дозволивши розробникам створювати клієнтські скрипти для взаємодії з користувачем без необхідності звертатися до сервера. Це дозволило створювати більш динамічні та реактивні вебсторінки. З часом JavaScript еволюціонував і тепер включає такі можливості, як асинхронне програмування, об’єктно-орієнтоване програмування, а також можливість використовувати його на серверній стороні, завдяки оточенню Node.js, що дозволило використовувати одну мову на всіх етапах розробки. Це прискорило процес розробки і знизило кількість можливих помилок, пов’язаних з використанням різних мов.

Також JavaScript – одна з найпопулярніших мов програмування, з великою активною спільнотою, яка створює багато ресурсів для навчання, туторіалів, бібліотек та інструментів. Крім того, JavaScript має широку

підтримку від сучасних веббраузерів, що дозволяє створювати більш динамічні та інтерактивні вебсторінки [25].

Отже, враховуючи всі ці фактори, можна зробити висновок, що JavaScript є найкращим вибором для розробки платформи для створення вебсторінок [26, 27]. Він надає всі необхідні інструменти та можливості для створення ефективного, гнучкого та масштабованого вебзастосунка.

Також не мало важливим є використання фреймворків для серверної та клієнтської частини застосунку. Фреймворки надають структуру та набір інструментів, які допомагають спростити та прискорити процес розробки.

Не використовуючи сучасні фреймворки, розробники можуть зіткнутися з рядом проблем, а саме написання коду з нуля може бути трудомістким і часомістким процесом, без використання фреймворків, може бути важко забезпечити консистентність та сумісність коду, також розробники мають самостійно вирішувати проблеми, які вже були вирішені в існуючих фреймворках, такі як маршрутизація, управління станом, взаємодія з базами даних і так далі.

Для розробки клієнтської частини було обрано найсучасніший фреймворк Solid.js. Solid.js – це високопродуктивний, реактивний фреймворк для JavaScript, який надає декларативні, компонентні абстракції для створення інтерфейсів користувача. Його реактивна модель базується на концепції «сигналів», що є функціями, які зберігають стан і автоматично сповіщають про зміни [28].

Сигнали – це ключовий компонент реактивної системи Solid.js. Кожен сигнал представляє собою джерело істини для певного значення, і коли це значення змінюється, сигнал автоматично повідомляє всіх слухачів про цю зміну. Це дозволяє створювати реактивні вирази, які автоматично оновлюються, коли змінюються їх залежності.

Solid.js використовує сучасну модель рендерінгу яка ґрунтується на концепції сигналів. Замість використання віртуального DOM, як у React та інкрементального оновлення Angular, Solid.js рендерить зміни безпосередньо

в реальний DOM, застосовуючи новітні методики оптимізації. Це означає, що він уникає витрат на віртуальний DOM та підвищує продуктивність. Що робить його найпродуктивнішим клієнтським фреймворком на даний момент.

Стан у Solid.js керується за допомогою сигналів і сторів. Сигнали дозволяють слідкувати за змінами значень, а стори надають більш високорівневу абстракцію для управління станом, що дозволяє використовувати такі речі, як асинхронні операції або залежність від інших сторів.

Важливо врахувати що Solid.js має дуже малий розмір бандла, що робить його ідеальним для використання в середовищах з обмеженим пропуском. Він також пропонує більшу гнучкість у порівнянні з React [29] та Angular [30], оскільки він не примушує вас до певної архітектури або патерну програмування.

Як і React, Solid.js використовує JSX. JSX – це синтаксичне розширення для JavaScript, яке дозволяє писати HTML-подібний синтаксис безпосередньо в JavaScript коді. Воно було вперше введено в React і зараз широко використовується в багатьох JavaScript бібліотеках і фреймворках, включаючи Solid.js. JSX допомагає зробити код більш зрозумілим і організованим, особливо коли створюється складний UI. Він дозволяє описувати структуру інтерфейсу користувача буквально в тій же манері, як ви б це робили у HTML. Це робить ваш код більш читабельним і зрозумілим, особливо для тих, хто вже знайомий з HTML.

Однак JSX – це не просто HTML всередині JavaScript. Він також дозволяє використовувати всю потужність JavaScript всередині розмітки, дозволяючи вставляти змінні, викликати функції і використовувати умовну логіку прямо в середині розмітки.

Використання JSX в Solid.js та React допомагає розробникам писати більш зрозумілий, організований та масштабований код. Він поєднує переваги декларативного синтаксису з потужністю JavaScript, створюючи ідеальний інструмент для розробки сучасних вебзастосунків [31].

На підставі всього вище сказаного, можна зробити висновок, що використання Solid.js у розробці платформи для створення вебсторінок буде хорошим вибором. Він надає інструменти, які дозволяють створювати ефективні, гнучкі, а саме головне високопродуктивні вебзастосунки.

Для розробки серверної частини [32] було обрано фреймворк Nest.js. Nest.js – це потужний фреймворк для серверного JavaScript, створений з використанням TypeScript [33] і використовує прогресивний JavaScript. Він поєднує елементи об’єктно-орієнтованого програмування, функціонального програмування і функціонального реактивного програмування.

Одна з ключових характеристик Nest.js – це його модульність. Він використовує модулі для організації коду, що дозволяє розробникам зосередитися на окремих аспектах їхніх застосунків. Кожен модуль містить різні частини застосунку, такі як контролери, сервіси, провайдери та інше [34].

Nest.js також пропонує вбудований модуль для роботи з базами даних, який забезпечує однорідний інтерфейс для роботи з різними базами даних. Це робить Nest.js дуже гнучким вибором для проєктів, які можуть вимагати різних типів баз даних.

Перейдемо до порівняння двох потужних серверних фреймворків – Nest.js та Koa. Обидва з них мають свої унікальні характеристики та переваги, які роблять їх цінними інструментами в руках розробників. Koa – це більш легкий та гнучкий фреймворк, який забезпечує більше контролю над серверними застосунком. Він був розроблений командою, що створила Express.js, і націлено на подолання деяких обмежень та слабких сторін Express.js, пропонуючи сучасний та гнучкий API. Втім, ця гнучкість може призвести до необхідності більше роботи з боку розробника, особливо при створенні складніших застосунків [35].

Напроти, Nest.js призначений для побудови масштабованих, надійних і легко супроводжуваних застосунків. Він має більш структурований підхід до організації коду та надає більш багатий набір функцій «з коробки». Це може

значно прискорити процес розробки та зменшити кількість коду, який потрібно написати.

Nest.js також включає інтеграцію з TypeScript, що забезпечує статичну типізацію та інші переваги TypeScript. Це значно покращує якість коду, спрощує його розуміння та полегшує налагодження.

Однією з ключових переваг Nest.js є його підтримка модульності та ін’єкції залежностей. Це означає, що розробник може легко ізолювати різні частини застосунку та забезпечити залежності там, де вони потрібні. Це може зробити код більш організованим, масштабованим та легким для тестування.

У підсумку, використання Nest.js для платформи створення вебсторінок є вдалим вибором. Його гнучкість, модульність, інтеграція з TypeScript та різні опції для роботи з базами даних роблять його високоефективним інструментом для розробки сучасних, масштабованих серверних застосунків.

3.2 Розробка клієнтської частини платформи

Процес розробки клієнтської частини платформи для створення вебсторінок є важливим етапом при розробці цього проєкту. Одним із найбільш вимогливих та значущих завдань цього етапу стала розробка візуального редактора.

Редактор, як основний інструмент для користувачів, має бути не тільки потужним, але й інтуїтивно зрозумілим. Перше, що потрібно було визначити, – це спосіб відображення блоків, з яких будується вебсторінка. Для цього була розроблена власна деревоподібна структура даних.

Ця структура даних зберігає різноманітну інформацію про кожний блок: його ім’я, унікальний ідентифікатор, унікальне ім’я CSS-класу, за допомогою якого застосовуються стилі, унікальний набір налаштувань для конкретного блоку, а також інформацію про вкладені в нього блоки.

Важливо зазначити, що не всі блоки можуть бути вкладені в інші. Для прикладу, в блоки зображень або типографіки не можна вкласти інші блоки. З іншого боку, контейнерні блоки, такі як колонки, можуть вміщувати будь-які інші блоки. Ця функціональність надає нам бажану гнучкість та варіативність при створенні вебсторінок.

На рисунку 3.1 можна побачити приклад цієї структури.

```
export const structure: Structure = {
  id: 'body',
  component: ComponentNames.BODY,
  children: [
    {
      component: ComponentNames.BLOCK,
      id: '_1w',
      className: '_1w',
      styles: {
        display: 'block',
        'flex-direction': 'column',
        width: 'auto',
        overflow: 'visible',
      },
      children: [
        {
          id: '_1fdddew',
          component: ComponentNames.BLOCK,
          className: '_1fdddew',
          children: [
            {
              id: 'hiIDasd-1as',
              component: ComponentNames.TYPOGRAPHY,
              className: 'Typographyfaf',
              props: {
                text: '<p>Typographyfaf</p>',
              },
            },
          ],
        },
```

Рисунок 3.1 – Приклад структури з якого складається вебсторінка

Для реалізації рендеру вебсторінки по власній структурі даних було використано концепцію динамічних компонентів Solid.js. Динамічний компонент – це компонент, що може бути змінений або замінений під час виконання. Це означає, що при ініціалізації сторінки візуального редактора рендерер рекурсивно ітерується по структурі блоків та динамічно рендерить

необхідні для вебсторінки блоки виходячи з назви компонента зазначеного в структурі.

Процес додавання нових блоків до вже існуючої структури вебсторінки реалізовано за допомогою технології «drag-and-drop». «Drag-and-drop», або перетягування, є зручним механізмом інтерфейсу користувача, який дозволяє користувачам вибирати візуальні об’єкти мишею, перетягувати їх до визначеного місця та відпускати, щоб здійснити певну дію.

У контексті платформи, панель інструментів, розташована зліва, надає користувачу можливість вибрати специфічний блок, який він бажає відобразити на вебсторінці. Користувач може перетягнути цей блок до центральної частини редактора, де його можна буде розмістити в потрібному місці. Власноручна реалізація технології «drag-and-drop» була важливою частиною процесу розробки.

Одним із викликів, з яким прийшлося зіткнутися, це створення та розміщення так званих «drop zones» – областей, до яких користувач може «скинути» вибраний блок. Це було проблемою, оскільки додаткові елементи, які використовуються при реалізації «drag-and-drop», не повинні змішуватися з контекстом елементів, що відображають вебсторінку. Це важливо, оскільки такі додаткові елементи можуть вплинути та спотворити стилі блоку, задані користувачем платформи.

Відповідно, було прийнято рішення, винести ці додаткові елементи «drag-and-drop» в окремий контекст та розмістити їх за допомогою абсолютного позиціонування CSS. Такий підхід забезпечує, що оригінальні стилі користувацьких блоків не будуть порушені в процесі редагування та перетягування. Таким чином, користувачі можуть вільно маніпулювати блоками на своїх вебсторінках, не турбуючись про можливі непередбачені зміни у візуальному оформленні.

Оптимізація оновлення стилів користувацьких блоків виявилася важливою частиною процесу розробки. При першій реалізації, стилі були прямо застосовані до блоків за допомогою атрибута «style». Втім, цей підхід

викликав додаткові оновлення в DOM, що є структурою даних, яка представляє вміст вебсторінки. Це було особливо помітно при зміні кольору блоку, коли оновлення DOM могло призвести до зниження продуктивності.

Для вирішення цієї проблеми, було прийнято рішення генерувати окремий CSS клас для кожного блоку і виконувати зміни в CSSOM, який є аналогом DOM для CSS. CSSOM представляє CSS стилі, які застосовані до вебсторінки, і дозволяє JavaScript маніпулювати стилями динамічно.

Це включало в себе розробку власного сервісу для більш ефективної взаємодії з CSSOM. За допомогою цього сервісу, появилась змога змінювати стилі блоку, не торкаючись DOM дерева. Замість цього, змінювалися стилі прямо в CSSOM. Це не тільки дозволило уникнути деяких проблем, пов’язаних з оновленням DOM, але й прискорило роботу платформи [31].

Особливо важливо відзначити, що це рішення виявилося вкрай ефективним. У зв’язку з тим, що стилі можуть швидко змінюватися в процесі редагування вебсторінки, здатність безпосередньо впливати на CSSOM без непотрібних оновлень DOM є ключовим моментом для підвищення продуктивності та забезпечення гладкого користувацького досвіду. Отже, незалежно від того, скільки раз користувач змінює кольори, шрифти або інші візуальні характеристики блоку, він отримує миттєвий відгук без запізнень або падіння продуктивності.

Також варто відзначити, що цей підхід допомагає уникнути потенційних труднощів, пов’язаних з прямим впливом на DOM. Це може включати все від непередбачуваних змін в структурі сторінки до проблем з продуктивністю, які можуть виникнути при спробах оновити великі об’єми даних в DOM. Замість того, цей підхід забезпечує більш стабільну, надійну та високопродуктивну роботу платформи.

Переходячи до взаємодії між клієнтською та серверною частинами платформи, було застосовано асинхронні JavaScript і XML (AJAX) запити. AJAX – це техніка, яка дозволяє вебзастосункам виконувати запити до сервера

і отримувати відповіді без необхідності перезавантажувати всю сторінку. Це робить досвід користувача більш плавним і ефективним.

Для реалізації AJAX запитів використовується бібліотека axios. Axios – це бібліотека заснована на клієнтському HTTP для браузера і Node.js. Вона пропонує інтерфейс, який легко використовувати, і включає в себе корисні функції, такі як перехоплювачі запитів і відповідей, які використовуються у проєкті.

Власні сервіси взаємодії допомагають керувати цими запитами, використовуючи специфічні для застосунку методи та конфігурації. Це допомагає забезпечити належну організованість, зберігаючи при цьому гнучкість у відповідь на різноманітні вимоги та потреби застосунку.

Важливим аспектом реалізації є використання axios інтерцепторів. Інтерцептори дозволяють втручатись в процес запиту або відповіді перед тим, як вони будуть оброблені, або впливати на помилки, перш ніж вони будуть відхилені. Платформа використовує інтерцептори для додавання токена авторизації до запитів авторизованого користувача без додаткових зусиль. Це забезпечує безперервну і безшовну авторизацію на всіх рівнях взаємодії між клієнтом і сервером.

Важливо зазначити, що користування інтерцепторами дозволяє розгорнути систему авторизації, яка не тільки ефективна, але і надійна. Кожен запит, що відправляється від авторизованого користувача, автоматично включає в себе необхідний токен, що гарантує, що запит буде належним чином оброблений сервером. Такий підхід є надзвичайно корисним для застосунків, які потребують безперервної взаємодії між клієнтом і сервером.

Цей підхід до AJAX запитів та використання axios також дозволяє легко обробляти помилки. Завдяки використанню інтерцепторів, платформа має змогу перехоплювати помилки, перш ніж вони дійдуть до основного коду застосунку, і відповідно реагувати на них. Це включає в себе збереження помилок для подальшого аналізу, а також надання корисного зворотнього зв’язку користувачам нашого застосунку.

3.3 Розробка серверної частини платформи

Розробка серверної частини платформи для створення вебсторінок є невід'ємною і критично важливою складовою цього проєкту. Одним з ключових аспектів цього етапу є реалізація авторизації та аутентифікації користувачів, які дозволяють підтвердити ідентичність користувачів і надати їм відповідні права доступу.

Авторизація і аутентифікація є центральними компонентами безпеки будь-якого вебзастосунку. Авторизація полягає в наданні користувачам дозволу на виконання певних дій, тоді як аутентифікація визначає, хто саме користується системою. Ці дві концепції є взаємопов'язаними і разом вони створюють захищену систему, що забезпечує контроль доступу.

Для реалізації цих механізмів використовується підхід, заснований на JWT токенах. JWT – це відкритий стандарт, який визначає спосіб безпечної передачі інформації між двома сторонами як JSON-об'єкт. Ця інформація може бути підтверджена і надійна, оскільки вона підписана. JWT використовуються для аутентифікації користувачів, і вони можуть містити необхідну інформацію про користувача для авторизації [36].

У контексті Nest.js, авторизація та аутентифікація з JWT реалізовані за допомогою модуля @nestjs/jwt. Цей модуль надає ряд служб та декораторів, які полегшують роботу з JWT. Для захисту різних маршрутів використовуються так звані «guards». Guards в Nest.js – це класи, що реалізують метод «canActivate», який вирішує, чи може запит продовжити своє виконання до маршруту. В контексті JWT, Guard може перевірити наявність токена в запиті, а також його дійсність. Якщо токен є і він дійсний, Guard дозволить продовжити запит до маршруту, в іншому випадку запит буде зупинено.

Один з власноруч реалізованих Guards – це JwtAuthGuard, який автоматично валідує JWT токен у заголовку авторизації запиту. Якщо токен відсутній або недійсний, запит буде зупинено.

Усі ці механізми дозволяю забезпечити надійну авторизацію та аутентифікацію користувачів у платформі для створення вебсторінок. Це надає можливість користувачам з впевненістю та безпекою використовувати наші послуги.

Також немаловажливим є взаємодія із базою даних. У контексті цього проєкту було використано TypeORM. TypeORM є одним з найпопулярніших ORM інструментів для TypeScript і JavaScript. Він надає абстракцію рівня даних, дозволяючи розробникам використовувати TypeScript класи як моделі, що спрощує взаємодію з базою даних.

В контексті Nest.js, TypeORM є натуральним вибором, оскільки Nest.js також використовує TypeScript як основну мову. Це означає, що є змога використання всіх переваг TypeScript, такі як статична типізація і інтелектуальне завершення коду, під час роботи з базою даних.

TypeORM надає також багатий набір функціональності, включаючи підтримку транзакцій, міграцій, вкладених вибірок, відношень між сутностями і багато іншого. Це допомагає полегшити роботу з базою даних і дозволяє зосередитися на бізнес-логіці застосунку, а не на деталях реалізації бази даних.

Використовуючи TypeORM в Nest.js, отримується перевага від інтеграції з фреймворком. Nest.js надає модуль TypeORM, який полегшує налаштування та використання TypeORM в рамках проєкту Nest.js. Він автоматично створює Singleton об’єкт підключення TypeORM, який можна використовувати в усьому застосунку. Модуль також надає декоратори для легкого внедрення репозиторіїв TypeORM в сервіси.

Однак, можливо, найбільша перевага використання TypeORM полягає в тому, що він дозволяє працювати на більш високому рівні абстракції, зосереджуючись на об’єктах у коді, а не на SQL запитах. Це полегшує розуміння та розробку застосунку.

Реалізація процесу публікації вебсайтів, створених користувачами платформи, стала вкрай важливим завданням в рамках даного проєкту. Цей процес передбачав надання можливості розміщення створених ресурсів у

глобальній мережі Інтернет. Для виконання цього завдання було орендовано вебсервер, а також був орендован домен, на якому мали розміщуватися як сама платформа, так і всі вебсторінки, створені користувачами цієї платформи.

Для забезпечення коректної роботи системи публікації було вирішено використовувати вебсервер nginx. Nginx – це високопродуктивний вебсервер з відкритим вихідним кодом, що використовується для обслуговування вебсторінок. Він відомий своєю стабільністю, багатим набором функцій, простотою конфігурації та низькими вимогами до ресурсів [37].

Встановлений та налаштований на орендованому вебсервері nginx слідкує за папкою ‘sites’, до якої додаються HTML-файли після публікації сайту користувачем. Це дозволяє автоматизувати процес публікації та забезпечити високу швидкість доступу до опублікованих сайтів.

Окрім того, nginx відповідає за створення сайтів на піддоменах, що вказані користувачем платформи. Це надає користувачам можливість розміщувати свої вебсайти на унікальних піддоменах, що сприяє підвищенню видимості їх проєктів в Інтернеті.

Правильне налаштування та оптимізація nginx не тільки підвищили продуктивність та надійність платформи, але і дозволили забезпечити гладке та швидке розгортання створених користувачами сайтів. Що є дуже важливим, оскільки одним з ключових факторів успіху будь-якого вебсайту є його доступність та швидкість завантаження, що безпосередньо впливає на досвід користувача.

В цілому, впровадження nginx у проєкт значно підвищило ефективність розгортання сайтів користувачів та дозволило створити масштабовану, гнучку та надійну інфраструктуру для платформи створення вебсторінок.

3.4 Тестування розробленої платформи

Під час тестування було протестовано ролі усіх прецедентів системи та їх функціонал. Сервіс має наступні ролі:

– незареєстрований користувач;

– неавторизований користувач;

– зареєстрований користувач.

Надалі буде розглянуто результати тестування сервісу для кожної з ролей.

Головна сторінка представлена для усіх користувачів однаково, та приведена на рисунку 3.2.

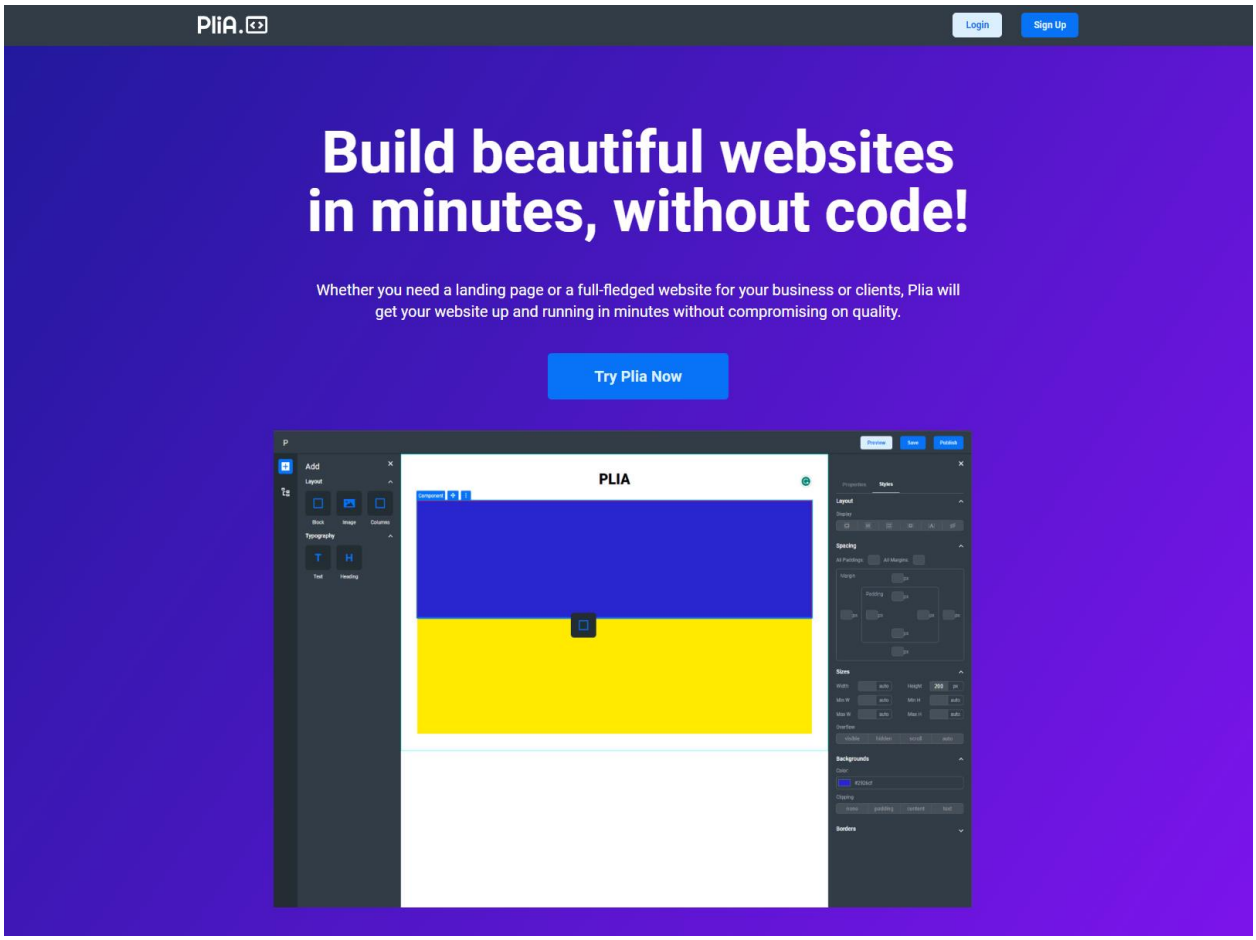

Рисунок 3.2 –Головна сторінка платформи

У подальшому, у вікні реєстрації, що зображено на рисунку 3.3, користувач, який ще не зареєстрований, має надати свою інформацію: ім’я, електронну адресу та пароль. Завершивши цей процес, буде здійснено створення аккаунту нового користувача на платформі [38].

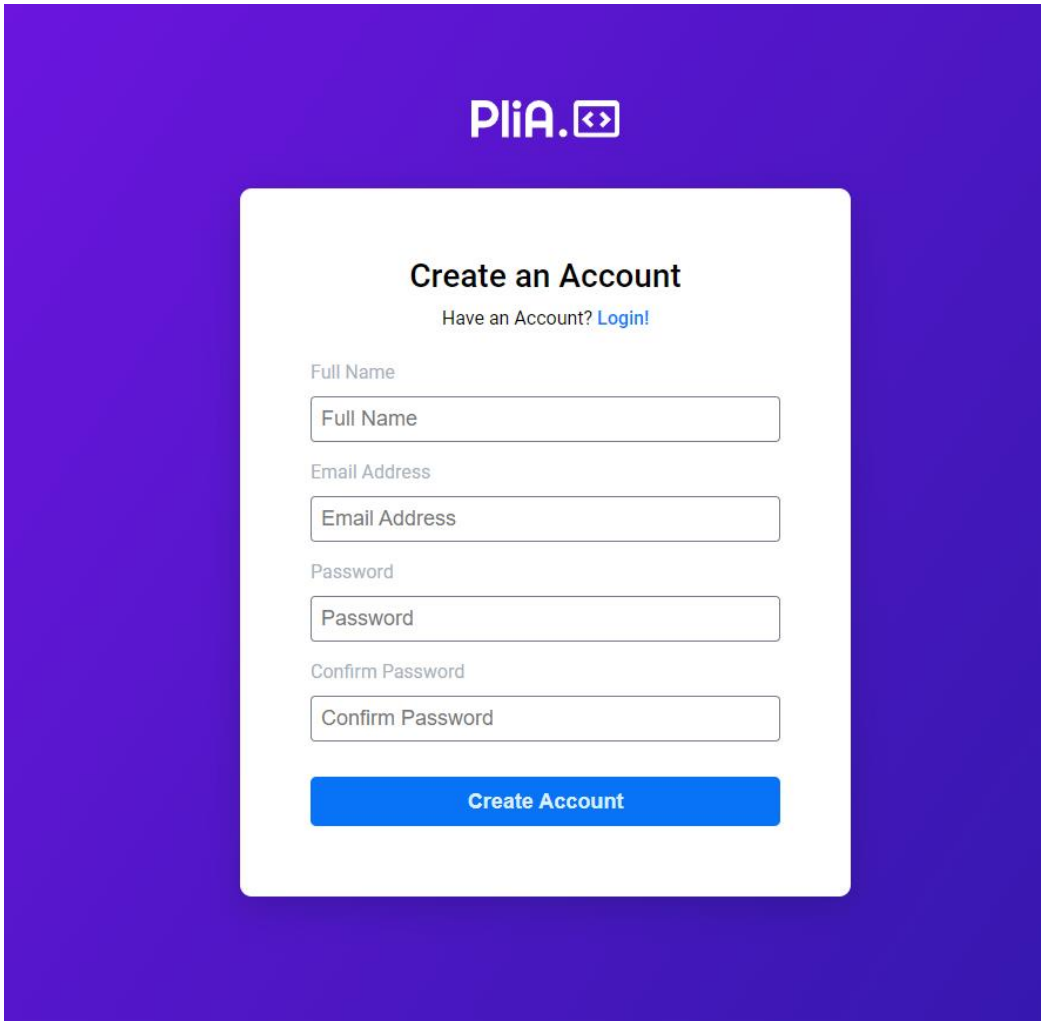

Рисунок 3.3 – Сторінка реєстрації платформи для створення вебсторінок

Не авторизований користувач може виконати авторизацію за допомогою кнопки у верхньому меню. Після цього йому буде доступне вікно, де слід ввести свої дані для авторизації – електронну пошту та актуальний пароль (рис. 3.4).

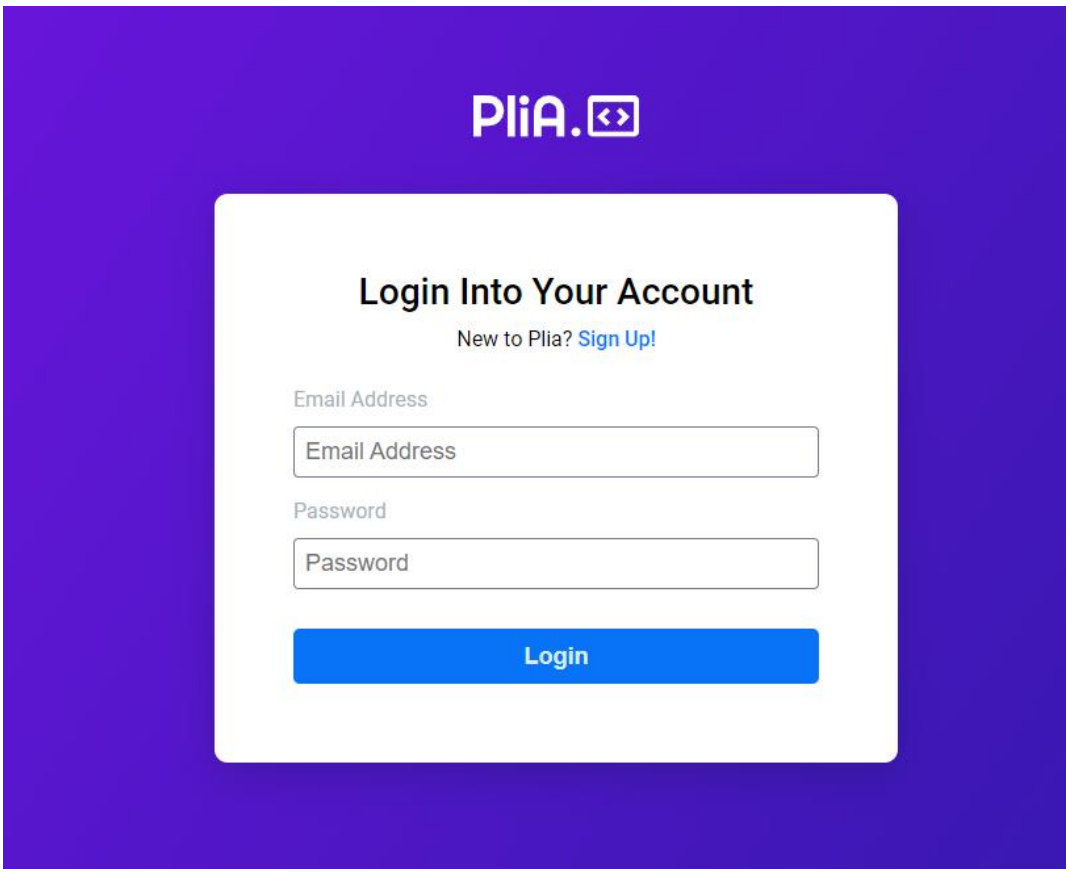

Рисунок 3.4 – Сторінка авторизації платформи для створення вебсторінок

Якщо дані про користувача будуть введені невірно, після натискання кнопки авторизації, з’явиться повідомлення про помилку (рис. 3.5).

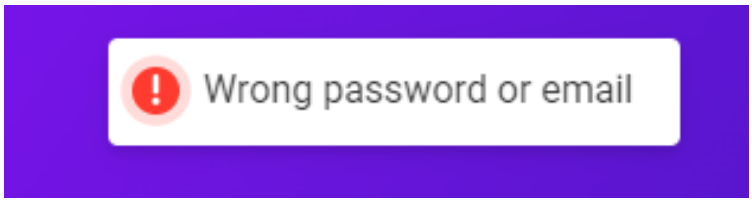

Рисунок 3.5 – Повідомлення про помилку при невірній авторизації на платформі

Коли реєстрація успішно завершена, користувач переходить до сторінки своїх вебсторінок і бачить повідомлення про те, що його обліковий запис було успішно створено. Оскільки на цей момент у новоприбулого користувача платформи ще немає жодної вебсторінки, екран відображає підказку про те, як створити нову вебсторінку (рис. 3.6).

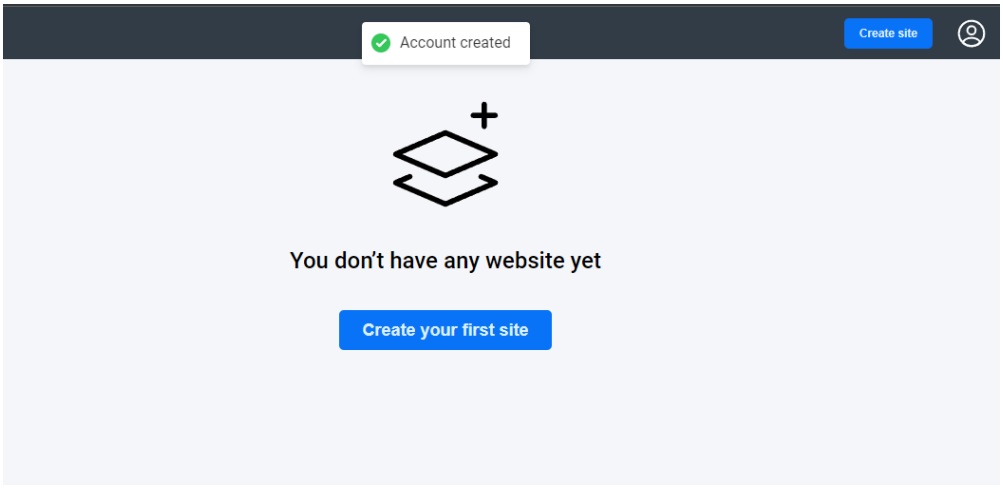

Рисунок 3.6 – Сторінка сайтів новоприбулого користувача

Після натискання на функціональну кнопку створення нового сайту. Користувача перенаправляє у візуальний редактор вебсторінок, де він може створити свою першу вебсторінку (рис. 3.7).

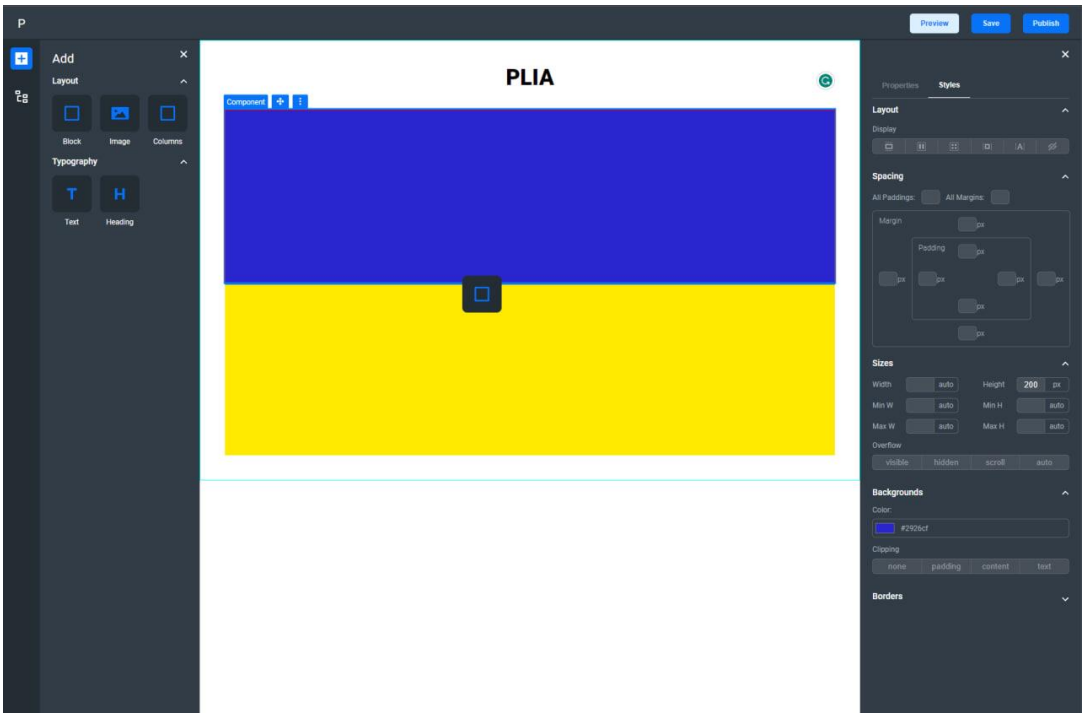

Рисунок 3.7 – Візуальний редактор платформи для створення вебсторінок

Закінчивши роботу над вебсайтом, користувач має можливість розмістити його в мережі. Він має зазначити назву свого проєкту та вибрати домен, на якому бажає розмістити свій сайт (рис. 3.8).

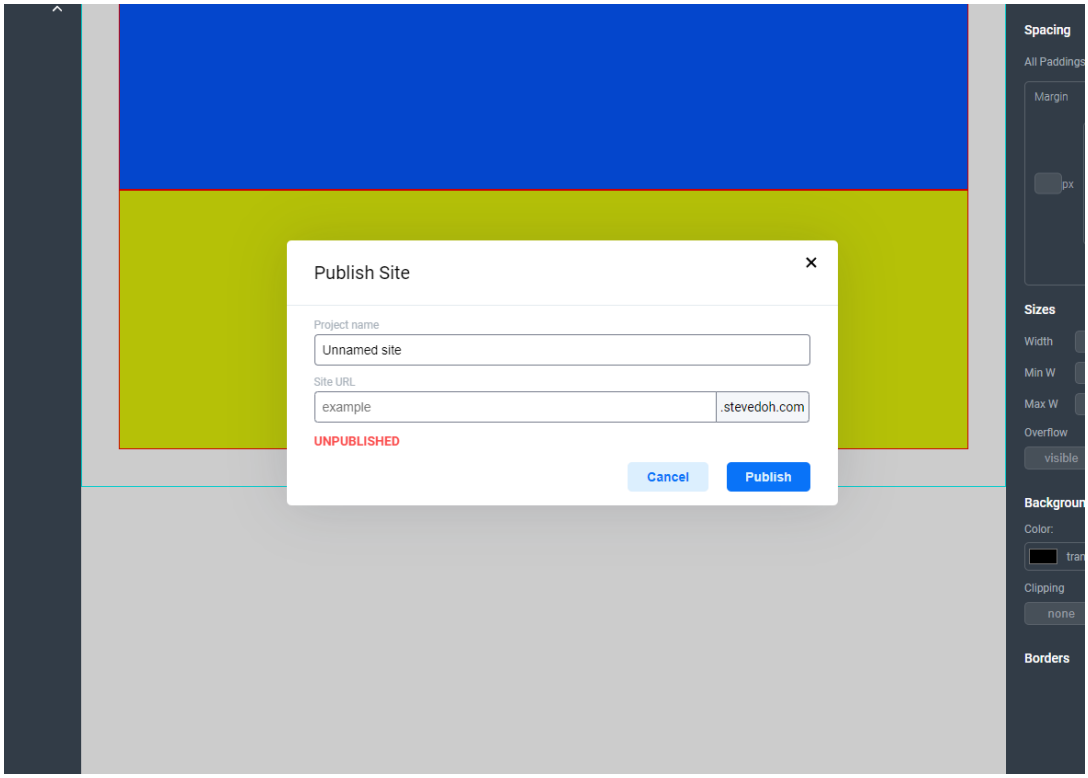

Рисунок 3.8 – Модальне вікно публікації сайту

Після введення всієї необхідної інформації та натискання кнопки «Publish» користувачу відображаються повідомлення про збереження та публікацію вебсайту (рис. 3.9). Далі, користувач має можливість перейти на вказаний домен, щоб побачити свій проєкт у мережі (рис. 3.10).

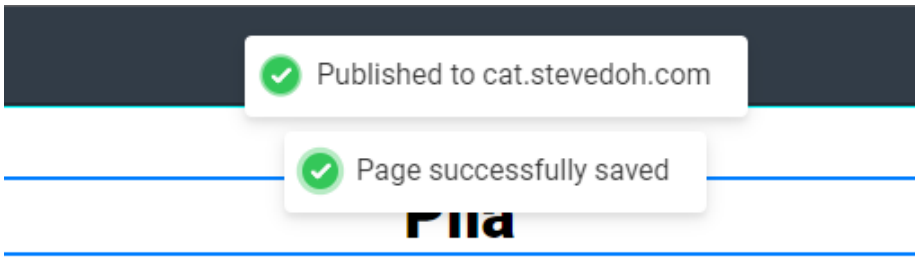

Рисунок 3.9 – Повідомлення про успішне створення та публікацію сайту

Рисунок 3.10 – Опублікований сайт користувача платформи

Тепер на сторінці користувача з вебсайтами відображається картка створеного проєкту. Тут він може перейти до візуального редактора, натиснувши кнопку «Edit Site», видалити проєкт, переглянути його або змінити налаштування (рис. 3.11).

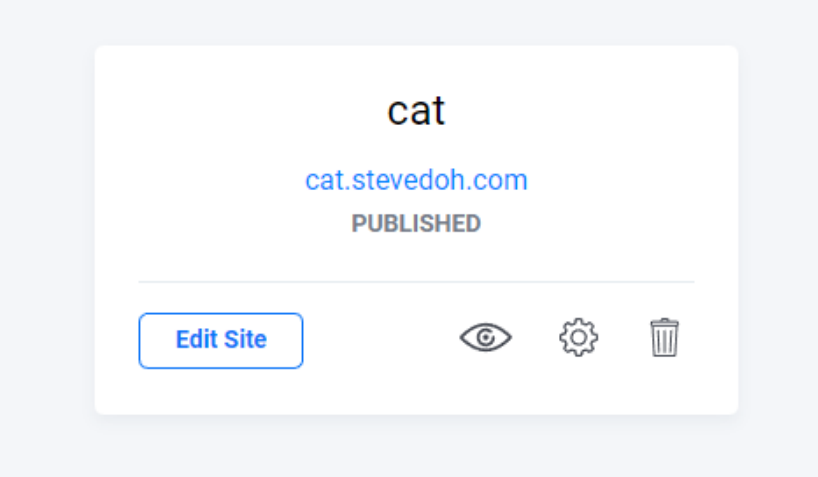

Рисунок 3.11 – Картка створеного сайту користувача платформи

Також зі сторінки своїх вебсайтів користувач має змогу перейти до сторінки налаштувань свого профілю або вийти з облікового запису, натиснувши на іконку вигляду людини у верхньому куті сторінки (рис. 3.12).

На сторінці профілю він має можливість змінити своє ім’я, електронну пошту або пароль (рис. 3.13).

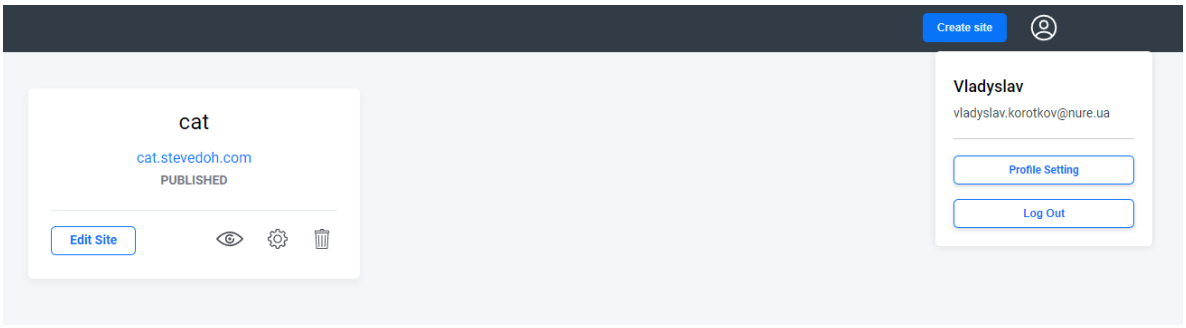

Рисунок 3.12 – Попап профілю на сторінці вебсайтів користувача

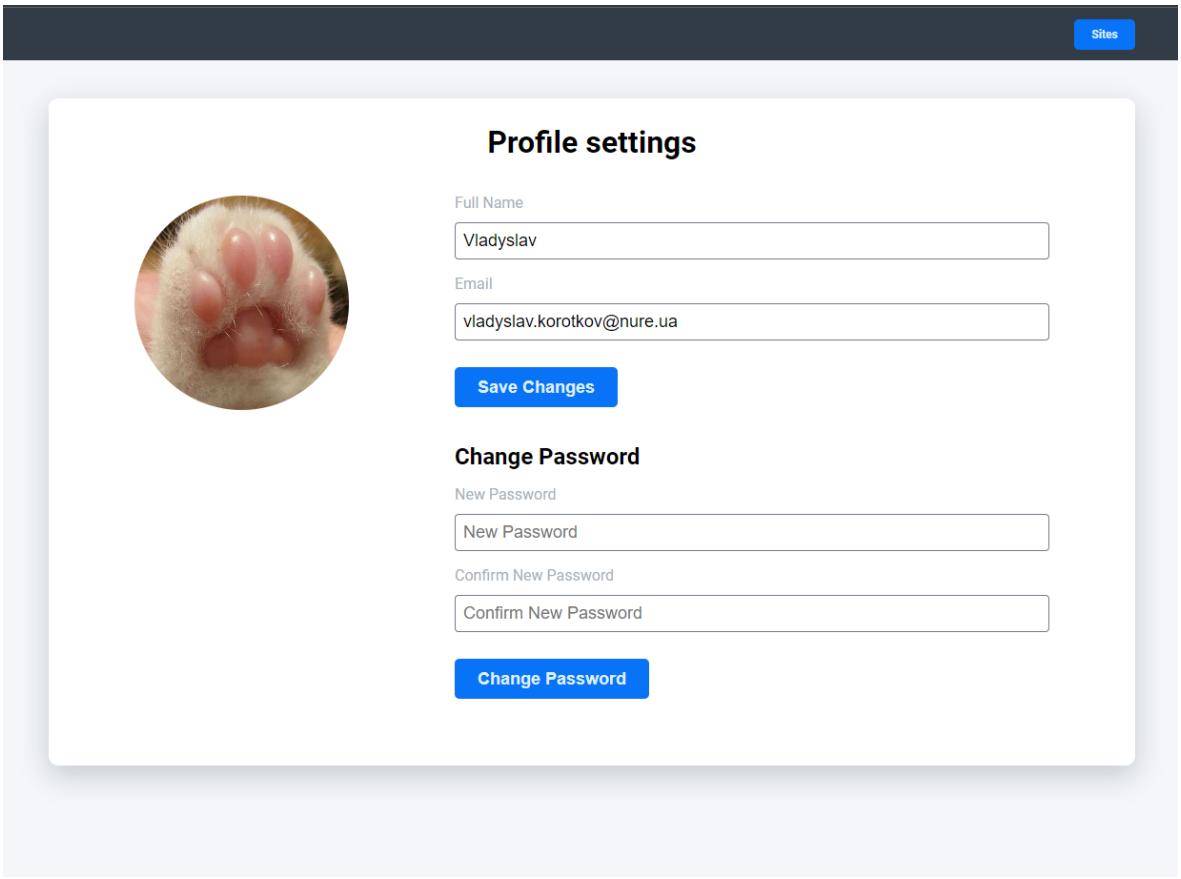

Рисунок 3.13 – Сторінка налаштувань профілю користувача

Тестування за усіма прецедентами системи було пройдено успішно, у системі наявних увесь функціонал, дефектів системи не було знайдено.

3.5 Перспективи подальшої роботи

Розроблена платформа містить мінімальний набір необхідних функцій для створення вебсторінок. Проте дана предметна область має широкий спектр додаткового функціоналу, яким можна поповнювати програмний застосунок.

Одним з ключових напрямків розвитку є інтеграція з дизайнерськими інструментами, такими як Figma. Figma – це платформа для співпраці в графічному дизайні, яка включає інструменти для створення дизайну інтерфейсу, прототипування та версіонування. Інтеграція платформи з Figma відкриє нові можливості для користувачів та спростить процес створення вебсторінок [39].

Використання API Figma дозволить користувачам імпортувати проєкти дизайну безпосередньо на платформу, перетворюючи дизайнерські елементи на відповідні блоки платформи для створення вебсторінок. Це значно зменшить час на розробку вебсторінки, оскільки не потрібно буде вручну перекладати дизайн у код.

Також гарною ідеєю було б використання штучного інтелекту для генерування вебсторінок на основі промптів. Промпт – це коротке повідомлення, яке надає користувач для наведення AI на те, що йому потрібно згенерувати. Наприклад, користувач може ввести «створити вебсторінку для продажу книг з темною темою і великим шрифтом», і AI сгенерує відповідну вебсторінку [40, 41].

Цей підхід відкриває цілий світ нових можливостей для автоматизації процесу розробки вебсторінок. Замість того, щоб вручну створювати кожен блок, користувачі зможуть просто описати своє бачення, а AI перетворить це на готову вебсторінку. Це не тільки значно спростить процес розробки для не

досвідчених користувачів, але і дозволить досвідченим розробникам швидко створювати прототипи та тестувати різні варіанти дизайну.

Також планується розширити набір доступних блоків та компонентів, дозволяючи користувачам створювати ще більш різноманітні вебсторінки. Зокрема планується додати більше блоків для електронної комерції, блогів, портфоліо.

Необхідно додати підтримку багатомовності до платформи, дозволяючи користувачам створювати вебсторінки на різних мовах без додаткових зусиль. Це дозволить користувачам досягти глобальної аудиторії та зробити свій контент доступним для більшого числа людей.

Подальший розвиток інтеграції з іншими сервісами та платформами також є важливим напрямком роботи. Планується розширити список підтримуваних сервісів для імпорту дизайну, інтеграції з системами управління контентом (CMS), системами електронної комерції та ін.

Даний функціонал, наведений вище, заплановано до реалізації на протязі наступних двох років. Для цього команда розробки повинна бути збільшена з одного розробника до трьох (як мінімум), та сьоми (як максимум). За песимістичного варіанту (три розробника) розробка системи буде виконуватися в форматі: два розробника відповідають за клієнтську частину платформи, інший за сереверну. В оптимістичному варіанті система матиме три фронтенд-розробника, два бекенд-розробника, тестувальника, та менеджера, який відповідатиме за зв’язок з клієнтами.

ВИСНОВКИ

У рамках кваліфікаційної роботи була розроблена і реалізована платформа для створення вебсторінок. Ця платформа включає в себе необхідний набір інструментів для швидкого та ефективного розроблення вебсторінок.

Основними напрямками роботи були:

– розробка системи авторизації та аутентифікації для користувачів платформи. Ця система дозволяє відслідковувати прогрес користувачів та забезпечує безпечний доступ до їх проєктів. Для цього було використано JWT токени, які надають можливість встановлювати та перевіряти ідентичність користувачів. Реалізація цього завдання була виконана з використанням технології Nest.js, яка забезпечує надійну та безпечну роботу з авторизацією та аутентифікацією;

– розробка візуального редактору вебсторінок з набором готових блоків, які можуть бути легко інтегровані та комбіновані у вебсторінку за допомогою «drag and drop» інтерфейсу. Це дозволяє користувачам створювати вебсторінки без необхідності написання коду. Кожен блок може бути вільно переміщений та налаштований згідно з потребами користувача;

– розробка системи публікації готових вебсторінок в інтернет на піддомені, що дозволить користувачам легко розгортати свої проєкти та надавати доступ до них іншим користувачам. Це значно спрощує процес випуску вебсторінок, що робить платформу більш привабливою для користувачів.

У процесі розробки було створено мінімально життєздатний продукт, який включає в себе всі ключові особливості, необхідні для ефективного створення та публікації вебсторінок. Використовуючи засоби MVP, користувачі можуть вже зараз ефективно використовувати платформу для реалізації своїх ідей і проєктів.

Важливо підкреслити, що в процесі розробки було приділено значну увагу оптимізації та підвищенню продуктивності. Використовуючи сучасні технології, такі як CSSOM, було досягнуто підвищення продуктивності платформи та поліпшення взаємодії користувачів з системою. Крім того, з допомогою TypeORM було значно поліпшено досвід роботи з базою даних.

Окрім того, було розроблено ряд сервісів для взаємодії між клієнтською та серверною частиною за допомогою AJAX запитів, які значно поліпшили взаємодію між різними частинами системи та полегшили розробку та тестування.

В подальшому планується продовження роботи над платформою, зокрема, над розробкою інтеграції з Figma для генерування вебсторінок на основі дизайну з цього сервісу. Також планується впровадження AI рішень для автоматичного генерування вебсторінок на основі вхідних промптів користувачів.

В цілому, можна зробити висновок, що розроблена платформа вже є повноцінним продуктом, який може бути використаний для швидкого та ефективного створення вебсторінок. Завдяки її гнучкості та адаптивності, платформа може бути легко адаптована під потреби різних користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Jacksi, K., & Abass, S. M. (2019). Development history of the world wide web. Int. J. Sci. Technol. Res, 8(9), 75-79.

2. Ndukwe, W. (2019). Website Builders: A Tool In Web Design From A Graphic Design Perspective.

3. Parker, J. M., James, K. W., & Ditt, C. (2022, June). Using Website Builders as a Tool for Teaching the Website Development Process: An Abstract. In Celebrating the Past and Future of Marketing and Discovery with Social Impact: 2021 AMS Virtual Annual Conference and World Marketing Congress (pp. 377-378). Cham: Springer International Publishing.

4. Finnegan, D. J., & Currie, W. L. (2010). A multi-layered approach to CRM implementation: An integration perspective. European Management Journal, 28(2), 153-167.

5. Pillai, A., Shinohara, K., & Tigwell, G. W. (2022, October). Website Builders Still Contribute To Inaccessible Web Design. In Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility (pp. 1-4).

6. Tilda. URL: https://tilda.cc/?lang=en (дата звернення 16.05.2023).

7. Rueda Villa, A. D. R. (2015). Wix, a free website builder as a teaching tool for the enhancement of extensive reading skills to students of first year bachillerato at unidad educativa PCEI Manuela Saenz, province of Santa Elena, 2015-2016 (Bachelor's thesis, La Libertad; Universidad Estatal Península de Santa Elena, 2015.).

8. Granqvist, R. (2022). The performance of Webflow: a comparative study.

9. Martin, R. C. (2017). Clean architecture.

10. Wu, Q., He, X., & Hintz, T. (2005, March). Bi-lateral filtering based edge detection on hexagonal architecture. In Proceedings.(ICASSP'05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005. (Vol. 2, pp. ii-713). IEEE.

11. Khalil, M. E., Ghani, K., & Khalil, W. (2016, April). Onion architecture: a new approach for xaas (every-thing-as-a service) based virtual collaborations. In 2016 13th Learning and Technology Conference (L&T) (pp. 1-7). IEEE.

12. 10 Most Popular Software Architectural Patterns. URL: https://nix-united.com/blog/10-common-software- architectural-patterns-part-1/ (дата звернення 20.05.2023).

13. Tvoroshenko, I., & Kharchenko, A. (2021). Some aspects of modern development for sign language recognition systems.

14. Tvoroshenko, I., & Babochkin, O. (2021). Object identification method based on image keypoint descriptors.

15. Гороховатський В.О., Творошенко І.С., Чмутов Ю.В. (2022). Застосування систем ортогональних функцій для формування простору ознак у методах класифікації зображень. Сучасні інформаційні системи, 6 (3), С. 5–12.

16. . Gorokhovatskyi V., Tvoroshenko I., Kobylin O., and Vlasenko N. (2023) Search for visual objects by request in the form of a cluster representation for the structural image description, Advances in Electrical and Electronic Engineering, 21(1), pp. 19-27.

17. Гороховатський В., Передрій О., Творошенко І., Марков Т. (2023). Матриця відстаней для множини компонентів структурного опису як інструмент для створення класифікатора зображень. Сучасні інформаційні системи, 7(1), С. 5-13.

18. Творошенко, І.С. (2018). Дослідження особливостей побудови нечітких відношень під час відображення динамічних взаємодіючих нечітких процесів складних систем.

19. Dey, T. (2011). A comparative analysis on modeling and implementing with MVC architecture. International Journal of Computer Applications, 1, 44-49.

20. Що таке база даних? URL: http://apeps.kpi.ua/shco-take-basa-danykh (дата звернення 20.05.2023).

21. Kinoshenko, D., Mashtalir, V., & Shlyakhov, V. (2007). A partition metric for clustering features analysis.

22. Brito, G., Terra, R., & Valente, M. T. (2018). Monorepos: a multivocal literature review. arXiv preprint arXiv:1810.09477.

23. Feature-Sliced Design. URL: https://feature-sliced.design/docs/get-started/overview (дата звернення 08.05.2023).

24. Jensen, S. H., Møller, A., & Thiemann, P. (2009, August). Type analysis for javascript. In SAS (Vol. 9, pp. 238-255).

25. Siahaan, M., & Vianto, V. O. (2022). Comparative Analysis Study of Front-End JavaScript Frameworks Performance Using Lighthouse Tool. Jurnal Mantik, 6(3), 2462-2468.

26. Tvoroshenko, I. (2020). Information technologies for decision-making on the conditions of spatially distributed objects. In I International Scientific and Practical Conference. Problems and perspectives of modern science and practice, Austria (pp. 45-50).

27. Творошенко, І.С. (2021). Технології прийняття рішень в інформаційних системах: навч. посібник. Харків: ХНУРЕ

28. Schuster, C., & Flanagan, C. (2016, March). Reactive programming with reactive variables. In Companion Proceedings of the 15th International Conference on Modularity (pp. 29-33).

29. Fedosejev, A. (2015). React. js essentials. Packt Publishing Ltd.

30. Geetha, G., Mittal, M., Prasad, K. M., & Ponsam, J. G. (2022, December). Interpretation and Analysis of Angular Framework. In 2022 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS) (pp. 1-6). IEEE.

31. Shroff, P. H., & Chaudhary, S. R. (2017, April). Critical rendering path optimizations to reduce the web page loading time. In 2017 2nd International Conference for Convergence in Technology (I2CT) (pp. 937-940). IEEE.

32. Chhetri, N. (2016). A comparative analysis of node. js (server-side javascript).

33. Bierman, G., Abadi, M., & Torgersen, M. (2014). Understanding typescript. In ECOOP 2014–Object-Oriented Programming: 28th European Conference, Uppsala, Sweden, July 28–August 1, 2014. Proceedings 28 (pp. 257-281). Springer Berlin Heidelberg.

34. Pham, A. D. (2020). Developing back-end of a web application with NestJS framework: Case: Integrify Oy's student management system.

35. Hahn, E. (2016). Express in Action: Writing, building, and testing Node. js applications. Simon and Schuster.

36. Chaturvedi, A. (2022, June). Comparison of Different Authentication Techniques and Steps to Implement Robust JWT Authentication. In 2022 7th International Conference on Communication and Electronics Systems (ICCES) (pp. 772-779). IEEE.

37. Reese, W. (2008). Nginx: the high-performance web server and reverse proxy. Linux Journal, 2008(173), 2.

38. Di Lucca, G. A., Fasolino, A. R., Faralli, F., & De Carlini, U. (2002, October). Testing web applications. In International Conference on Software Maintenance, 2002. Proceedings. (pp. 310-319). IEEE.

39. Frick, A. (2022). From Design to Code: A Study on Generating Production Code From User Interface Design Software.

40. Творошенко, І.С. (2018). Особливості застосування сучасних принципів штучного інтелекту до розробки ефективних механізмів моделювання складних систем. Science and Technology of the Present Time: Priority Development Directions of Ukraine and Poland, 118-121.

41. Kinoshenko, D., Mashtalir, S., Stephan, A., & Vinarski, V. (1993). Neural Network Segmentation Of Video Via Time Series Analysis. INFORMATION THEORIES & APPLICATIONS, 232.