МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «УНІВЕРСИТЕТ «КРОК»

Фаховий коледж Університету «КРОК»

ДИПЛОМНА РОБОТА

за темою

«Розробка веб-сайтів з використанням JavaScript-бібліотек та фреймворків»

Студент 4 курсу групи КН-20	Керівник дипломної роботи
	викладач
	(посада керівника)
Клокун Олексій Олександрович	Чернозубкін Ігор Олександрович
(прізвище, ім'я та по-батькові студента)	(прізвище, ім'я та по-батькові керівника)
	До захисту
	(резолюція «До захисту»)
(підпис студента)	10.06.24 (дата) (підпис викладача)

Київ, 2024 рік

ПЕРЕЛІК ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

HTML (HyperText Markup Language)	стандартна мова розмітки для документів, призначених для відображення в веб-браузері
CSS (Cascading Style Sheets)	мова таблиці стилів, що використовується для визначення презентації та стилізації документа, написаного мовою розмітки, такою як HTML або XML
DOM (Document Object Model)	програмний інтерфейс для веб-документів
PWA (Progressive Web App.	вебзастосунок, який є гібридом звичайної вебсторінки та мобільного застосунку
AJAX (Asynchronous JavaScript and XML)	програмна техніка, яка використовує JavaScript і об'єкт XMLHTTPRequest для обміну даними між веб-браузером і веб-сервером
OWASP (Open Web Application Security Project)	онлайн-спільнота, яка створює вільно доступні статті, методології, документацію, інструменти та технології в галузі безпеки вебзастосунків
Front-end	публічна частина веб-застосунків (веб-сайтів), з якими користувач може безпосередньо взаємодіяти.
Back-end	розробка бізнес-логіки продукту (сайту або веб-застосунку)
MVC (Model-View-Controller)	шаблон проектування програмного забезпечення, який зазвичай використовується для розробки користувацьких інтерфейсів, які ділять пов'язану програмну логіку на три взаємопов'язані елементи

ЗМІСТ

ВСТУП

У сучасному світі веб-сайти стають все більш невід'ємною частиною нашого життя. За даними Statista за 2023 рік, 81,24% всієї інформації в Інтернеті передається через веб-трафік, і цей показник постійно зростає, свідчачи про постійно зростаючу потребу у якісних та динамічних веб-сайтах [1].

JavaScript відіграє ключову роль у створенні інтерактивних та динамічних веб-сайтів. За даними State of JavaScript 2023, 95,2% розробників веб-сайтів використовують JavaScript у своїй роботі [2]. Ця мова програмування є невід'ємною складовою для надання користувачам зручного та привабливого інтерфейсу. Допомагаючи створити інтерактивні елементи та відстежувати події, JavaScript забезпечує веб-сайти життєздатністю та функціональністю. Крім того, JavaScript-бібліотеки та фреймворки стали невід'ємною частиною процесу розробки, сприяючи прискоренню цього процесу та поліпшенню якості коду, що допомагає зменшити час і кількість помилок.

Згідно з MDN Web Docs JavaScript Usage Statistics, у 2023 році 93,8% веб-сайтів використовують принаймні одну JavaScript-бібліотеку [3].

Мета цього дослідження полягає у глибокому аналізі JavaScript-бібліотек та фреймворків, які застосовуються для створення веб-сайтів. Основний акцент буде зроблено на наступних питаннях: специфіка та особливості найпопулярніших інструментів у сучасній веб-розробці, порівняння їх функціональних можливостей, а також виявлення найкращих практик щодо вибору відповідного інструменту для конкретного проекту.

Детальна аналіз та порівняння допоможуть з'ясувати переваги та недоліки кожного інструменту, а також встановити критерії для їх вибору в реальних проектах. Практична частина дослідження передбачає розробку веб-сайту або

веб-додатку з використанням обраного інструменту, що дозволить перевірити ефективність та зручність його використання в реальних умовах.

Результати цього дослідження мають на меті просунути розуміння переваг і недоліків різних інструментів для розробки веб-сайтів, а також сприяти їхньому практичному використанню. Методика дослідження буде включати аналіз літератури, збір та обробку даних про використання інструментів, а також експериментальну розробку та аналіз результатів.

Основні завдання дослідження:

систематизувати та узагальнити інформацію про найпопулярніші JavaScript-бібліотеки та фреймворки, які використовуються для створення веб-сайтів;

провести порівняльний аналіз функціональних можливостей, переваг та недоліків кожного інструменту;

визначити кращі практики щодо вибору відповідної бібліотеки або фреймворку для конкретного проекту;

продемонструвати практичне використання обраного інструменту на прикладі розробки веб-сайту або веб-застосунку;

сформулювати рекомендації щодо ефективного використання JavaScript-бібліотек та фреймворків для розробників веб-сайтів.

Практичне значення дослідження виявляється в контексті стрімкого розвитку веб-технологій та наростаючої складності веб-сайтів. Потреба у швидкому реагуванні на зміни та використанні ефективних інструментів для досягнення цілей проекту створює попит на дослідження, спрямоване на порівняння та вибір найкращих практик у веб-розробці з використанням JavaScript-бібліотек та фреймворків.

Методика дослідження. Порівняння та аналіз JavaScript-бібліотек та фреймворків, які використовуються для розробки веб-сайтів. Опис основних принципів та класифікації цих інструментів, а також їх впливу на

користувацький досвід та процес розробки. Дослідження не розглядає альтернативні технології, обмежуючись виключно JavaScript-екосистемою.

Структура роботи. Звіт складається з трьох розділів, п'яті додатків, список використаних джерел вміщує 31 найменувань, 92 сторінки тексту.

Перший розділ описує історію та сучасні тенденції веб-розробки, а також роль JavaScript у цьому контексті. Другий розділ – використання React для розробки конкретного типу веб-сайту. Третій розділ присвячений практичному впровадженню JavaScript-бібліотек та фреймворків через розробку інтерактивного інтерфейсу веб-сайту.

РОЗДІЛ 1 ІСТОРІЯ ТА СУЧАСНІ ТЕНДЕНЦІЇ ВЕБ-РОЗРОБКИ

1.1. Історія розвитку веб-розробки: Від початків до сучасності

Основою front-end розробки є потужне тріо HyperText Markup Language (HTML), Cascading Style Sheets (CSS) і JavaScript. Кожна мова відіграє визначну, але взаємопов'язану роль у формуванні користувацького досвіду Всесвітньої павутини.

HyperText Markup Language (HTML), задумана Тімом Бернерсом-Лі в кінці 1980-х років, слугує основним шаром [4]. Вона визначає структуру та зміст веб-сторінки за допомогою набору тегів, що вказують на елементи, такі як заголовки, абзаци та зображення. Простота та легкість використання HTML дозволили швидко створювати веб-сторінки, сприяючи початковому зростанню Інтернету. Однак її статичний характер обмежував можливість створення візуально привабливих та інтерактивних досвідів.

Cascading Style Sheets (CSS), представлені наприкінці 1990-х років, вирішили обмеження HTML, надаючи механізм для відокремлення змісту від презентації [4]. CSS дозволяє розробникам визначати стилі для різних елементів HTML, контролюючи такі аспекти, як макет, шрифти та кольори. Це розділення відповідальностей не лише поліпшує підтримку коду, але й дозволяє розробникам створювати візуально привабливі та послідовні веб-досвіди.

Поява JavaScript наприкінці 1990-х років відзначила поворотний момент. Ця скриптова мова надає веб-сторінкам інтерактивність та динамічність [3]. На відміну від HTML та CSS, JavaScript дозволяє розробникам створювати динамічні поведінки, такі як реакція на введення користувача, маніпулювання Об'єктною Моделлю Документа (DOM) та асинхронний обмін даними з серверами. Універсальність JavaScript була інструментальною в еволюції веб-додатків, дозволяючи створювати багаті користувацькі досвіди, які раніше були неможливими в чисто статичному середовищі.

Взаємодія між HTML, CSS та JavaScript є фундаментальною для сучасної веб-розробки. HTML визначає структуру, CSS визначає презентацію, а JavaScript надає життя веб-сторінці. Ця взаємодія дозволяє створювати інтерактивні та динамічні веб-досвіди, з якими ми стикаємося сьогодні. Майбутні досягнення в цих основних технологіях, спільно з появою нових фреймворків та бібліотек, обіцяють подальше розширення можливостей в мережі.

Еволюція Всесвітньої Павутини була відзначена фундаментальним зміщенням - переходом від статичних веб-сторінок до динамічних інтерфейсів. Початкові веб-сторінки, побудовані переважно з використанням HTML, функціонували як цифрові буклети, пропонуючи обмежену взаємодію з користувачем та відображення інформації. Однак поява мов скриптів на серверному боці та технологій на клієнтському боці, таких як JavaScript, відкрила шлях до більш цікавого та взаємо дійного веб-досвіду.

Обмеження статичного HTML були очевидними. Оновлення контенту вимагали ручних змін у коді, що ускладнювало доставку інформації в реальному часі та персоналізацію [5]. Мови скриптів на серверному боці, такі як PHP, ASP та Python, пропонували вирішення. Ці мови дозволяли генерувати динамічний контент на основі введення користувача або маніпуляції даними на сервері. Наприклад, функціонал пошуку продуктів на веб-сайті електронної комерції використовував би скрипти на серверному боці для отримання та відображення відповідної інформації про продукти на основі запитів користувача.

Введення JavaScript наприкінці 1990-х років ще більше революціонізувало веб-інтерфейси. На відміну від скриптів на серверному боці, JavaScript виконує код безпосередньо в браузері користувача, дозволяючи маніпулювати контентом в реальному часі та взаємодіяти з користувачем. Здатність JavaScript взаємодіяти з (DOM) веб-сторінки дозволила використовувати динамічні ефекти, такі як перевірка форм, каруселі зображень

та інтерактивні анімації [5]. Цей перехід до обробки на клієнтському боці не лише покращив час завантаження сторінок, але й сприяв більш реагуючому та захопливому користувацькому досвіду.

Поява фреймворків і бібліотек, побудованих на цих основних технологіях, ще більше прискорила розвиток динамічних веб-інтерфейсів. Фреймворки, такі як Ruby on Rails та Django, надали готовий функціонал та структури, спрощуючи розробку веб-додатків та сприяючи пере використанню коду. Ці досягнення дозволили розробникам створювати складні, інтерактивні веб-додатки, які конкурують зі стаціонарним програмним забезпеченням за функціональністю.

Перехід від статичних веб-сторінок до динамічних інтерфейсів докорінно змінив спосіб взаємодії з інтернетом. Це відкрило епоху персоналізованого досвіду, доступу до даних у реальному часі та багатого залучення користувачів. Забігаючи наперед, досягнення в області веб-технологій, таких як WebAssembly і прогресивні веб-додатки (PWA), обіцяють ще більш захоплюючий і динамічний веб-досвід, розмиваючи лінії між веб-додатками і нативними додатками.

Початок 21-го століття став свідком парадигматичної зміни у веб-розробці з виникненням нової технологічної триади: Асинхронний JavaScript і XML (AJAX), Об'єктна Нотація JavaScript (JSON) та веб-сервіси. Ця сполученість революціонізувала користувацький досвід, відкривши еру Вебу 2.0 - вебу, що характеризується контентом, згенерованим користувачами, динамічними взаємодіями та обміном даними в реальному часі.

AJAX, введений наприкінці 2000-х років, фундаментально змінив спосіб взаємодії веб-сторінок з серверами. Традиційно дія користувача спричиняла повне оновлення сторінки, знову завантажуючи весь контент. Однак AJAX дозволяв асинхронну комунікацію. JavaScript у браузері міг робити цільові запити на сервер, оновлюючи конкретні частини сторінки без повного

перезавантаження. Це призвело до більш плавного та реагуючого користувацького досвіду, схожого на поведінку десктопних програм [6].

Проте ефективний транспорт даних між браузером та сервером потребував легкого та мово-незалежного формату. З'явився JSON, людино-читабельний текстовий формат для структурування даних. Порівняно з його попередником, XML, JSON пропонував простіший та швидший підхід до обміну даними, ідеально відповідаючи вимогам AJAX-комунікації [7].

Веб-сервіси забезпечили основу для цієї динамічної взаємодії. Зазвичай, це програмні застосунки, доступні через Інтернет, що дотримуються стандартних протоколів, таких як SOAP або REST. Використовуючи веб-сервіси для виклику функціоналу, розробники могли створювати модульні та повторно використовувані компоненти, що дозволяли веб-додаткам спілкуватися та обмінюватися даними безперешкодно [8].

Взаємодія цих технологій фундаментально перетворила веб-ландшафт. Приклади застосування AJAX, такі як Gmail та Google Maps, стали всюди наявними, пропонуючи більш багатий та інтерактивний користувацький досвід. Прийняття JSON як стандарту для обміну даними сприяло плавній комунікації між різноманітними додатками та платформами. Веб-сервіси, з іншого боку, дозволили створювати складні та взаємопов'язані веб-екосистеми.

У висновку, поява AJAX, JSON та веб-сервісів відзначила переломний момент у веб-розробці. Їх синергетичні відносини перетворили веб зі статичної платформи для розповсюдження інформації в динамічне та інтерактивне середовище, що продовжує еволюціонувати та впливати на спосіб, яким ми взаємодіємо з цифровим світом сьогодні.

1.2. Сучасні технології веб-розробки: Огляд і новітні підходи

Сучасний ландшафт розвитку фронтенду обертається навколо потужної триади: HTML5, CSS3 та JavaScript-фреймворків, таких як Angular, React та Vue.js. Ця комбінація дозволяє розробникам створювати багаті, інтерактивні та візуально захоплюючі інтерфейси користувача (UI) для веб-додатків.

HTML5, камінь кутовий структури веб-сторінок, пропонує значні покращення порівняно зі своїм попередником. Він вводить семантичні елементи, які надають більше значення контенту, сприяючи кращій доступності та оптимізації для пошукових систем (SEO). Крім того, HTML5 включає мультимедійні можливості, що дозволяють вбудовувати елементи аудіо, відео та полотна безпосередньо в веб-сторінки, виключаючи необхідність зовнішніх плагінів [9].

CSS3, мова для стилізації та презентації, пройшла помітну еволюцію. Вона надає безліч функцій для створення динамічних та анімованих UI без великого використання JavaScript. Такі функції, як градієнти, переходи та трансформації, дозволяють розробникам створювати візуально привабливий та чуйний досвід, який легко адаптується до різних розмірів екрану та пристроїв [10].

Проте, справжня сила сучасного розвитку фронтенду полягає в використанні JavaScript-фреймворків. Ці фреймворки надають готові структури та функціональні можливості, спрощуючи процес розробки. Хоча кожна структура має свої сильні та слабкі сторони, виникають деякі ключові міркування.

Angular, розроблений і підтримуваний Google, є комплексним фреймворком, ідеальним для створення великомасштабних одно сторінкових додатків (SPAs). Його структурований підхід забезпечує чітке розділення проблем, сприяючи підтриманню коду. Однак його жорсткість може становити

виклик для менших проектів або розробників, які тільки починають з JavaScript [11].

React, створений Facebook, є універсальною бібліотекою, яка акцентується на розвитку на основі компонентів. Цей модульний підхід дозволяє повторне використання коду та ефективне управління складними UI. Гнучкість React робить його підходящим для проектів різних масштабів, але його залежність від зовнішніх бібліотек може ускладнити роботу для початківців [12].

Vue.js, ідея колишнього розробника Google, пропонує баланс між структурою Angular та гнучкістю React. Він має простіший шлях вивчення, що робить його привабливим вибором для початківців. Прогресивний характер Vue.js дозволяє поступове впровадження в існуючих проектах, що подальше збільшує його привабливість [13].

На завершення, HTML5, CSS3 та JavaScript-фреймворки стали невід'ємними інструментами для сучасного розвитку фронтенду. Їх спільна сила дозволяє створювати інтерактивні, візуально захоплюючі та користувацький дружні веб-додатки, які продовжують розширювати межі цифрового досвіду. Хоча вибір фреймворка залежить від вимог проекту та експертизи розробника, ці технології колективно представляють основу для будівництва вебу завтрашнього дня.

Світ back-end розробки процвітає на міцній основі, яку створює обраний групою мов програмування: Node.js, Python, PHP та Java. Кожна мова пропонує власні переваги та відповідає конкретним вимогам проекту, формуючи складну логіку та функціональність, яка дозволяє працювати веб-додаткам.

Node.js, побудований на популярному движку JavaScript Chrome V8, виділяється своєю асинхронною, подійно-орієнтованою архітектурою. Він робить його особливо майстерним в обробці додатків в реальному часі з високою паралельністю, таких як чат-додатки або платформи соціальних

мереж. Крім того, Node.js сприяє спрощенню процесу розробки, дозволяючи розробникам використовувати JavaScript як для логіки фронтенду, так і для логіки back-end, зменшуючи тертю при зміні контексту [14].

Python, відомий своєю зрозумілістю та обширними бібліотеками, відмінно підходить для швидкого прототипування та застосувань в галузі науки про дані. Його чіткий синтаксис та великий екосистема пакетів роблять його популярним вибором для проектів, що потребують складної обробки даних чи інтеграції машинного навчання. Однак динамічна типізація Python іноді може призводити до помилок в час виконання, що може бути проблемою для додатків з високими вимогами до продуктивності [15].

PHP, ветеран сцени веб-розробки, залишається визначним гравцем завдяки великій спільноті, зрілим фреймворкам, таким як Laravel, і легкості розгортання на платформах спільного хостингу. Його синтаксис має схожості з C, що робить його доступним для розробників з досвідом у цих мовах. Хоча PHP не є вбудовано об'єктно-орієнтованим, покращення в пізніших версіях вирішують цей недолік в певній мірі [16].

Java, корпоративний трудівник, пропонує неперевершену стабільність, масштабованість та безпеку. Його статична типізація забезпечує безпеку типів, мінімізуючи помилки в час виконання та роблячи його ідеальним для великих масштабів, критичних застосунків у фінансовому чи медичному секторах. Проте код Java може бути масштабним порівняно з більш лаконічними мовами, що може вплинути на швидкість розробки [17].

Оптимальний вибір серед цих мов залежить від конкретних потреб проекту. Node.js блищить для додатків у реальному часі, Python відмінний у науці про дані та швидкому розвитку, PHP залишається надійним вибором для веб-додатків з фокусом на легкість розгортання, а Java володіє бездоганною репутацією для великих, безпечних корпоративних додатків.

На завершення, Node.js, Python, PHP та Java представляють основу сучасного back-end розвитку. Розуміння їх переваг та недоліків дозволяє розробникам вибирати найбільш підходящу мову для створення міцних, масштабованих та ефективних веб-додатків, які відповідають різноманітним потребам користувачів.

Цифровий пейзаж ґрунтується на критичній основі: базах даних. У сфері веб-розробки вибиваються три провідних гравці – MySQL, PostgreSQL та MongoDB – кожен з яких пропонує унікальні переваги й задовольняє різноманітні потреби у керуванні даними.

MySQL, широко використовувана відкрита реляційна система управління базами даних (RDBMS), славиться своєю легкістю використання, швидкістю та надійністю. Її структурований підхід, заснований на таблицях з рядками та стовпцями, що забезпечують визначені відносини, робить її ідеальною для зберігання та управління чітко визначеними, взаємопов'язаними наборами даних. Популярність MySQL додатково підтримується її сумісністю з різними мовами програмування та обширною спільнотою підтримки. Однак його жорсткість схеми може стати передачею для еволюції моделей даних, а масштабованість для величезних наборів даних може бути обмеженою [18].

PostgreSQL, ще одна відкрита RDBMS, пропонує привабливу альтернативу. Вона має потужний набір функцій, включаючи підтримку складних типів даних, транзакції ACID (забезпечуючи цілісність даних) та розширене управління користувачами. Об'єктно-реляційні можливості PostgreSQL забезпечують більшу гнучкість у моделюванні даних порівняно з MySQL. Крім того, його виняткова масштабованість робить його сильним конкурентом для обробки великих і ростучих наборів даних. Однак багатство функцій може мати свою ціну у вигляді складності вивчення порівняно з MySQL [19].

MongoDB, NoSQL (нереляційна) база даних, стоїть у різкому контрасті з вищезгаданими рішеннями RDBMS. Вона використовує модель даних на основі документів, зберігаючи дані у гнучких структурах, схожих на JSON. Цей підхід без схеми пропонує виняткову гнучкість для зберігання та управління неструктурованими чи напівструктурованими даними, що робить його ідеальним для систем управління контентом, додатків у реальному часі та великих обсягів даних. Однак відсутність обов'язкових відносин схем може потребувати додаткових зусиль розробки для забезпечення консистентності даних [20].

Вибір оптимальної бази даних залежить від конкретних вимог проекту. Для структурованих даних з визначеними відносинами MySQL або PostgreSQL можуть бути ідеальними виборами. Однак для еволюції моделей даних чи проектів, які включають великі та неструктуровані набори даних, MongoDB пропонує привабливу альтернативу.

На завершення, MySQL, PostgreSQL та MongoDB представляють основу сучасного управління базами даних у веб-розробці. Розуміння їх переваг та недоліків дозволяє розробникам вибирати найбільш підходяще рішення для забезпечення цілісності та доступності даних, які стимулюють сучасні веб-додатки.

Сучасний ландшафт розробки програмного забезпечення прагне за потужним тріумвіратом: Git, Scrum та Agile-методології. Git, система контролю версій, надає розробникам можливість відстежувати зміни, ефективно співпрацювати та легко повертатися до попередніх версій.

Git, розподілена система контролю версій (DVCS), має значну перевагу перед традиційними централізованими системами. Вона дозволяє розробникам зберігати повну копію кодової бази на їхніх локальних машинах. Це дозволяє роботу в автономному режимі, сприяє співпраці через гілки та злиття, та забезпечує надійний захист з можливістю повернутися до будь-якої

попередньої версії коду. Інтуїтивний інтерфейс командного рядка Git та безліч графічних інтерфейсів користувача (GUIs) роблять його універсальним інструментом для розробників будь-якого рівня досвіду [21].

Scrum, популярна Agile-методологія, наголошує на ітеративному розвитку та постійному вдосконаленні. Вона працює в межах фіксованого періоду часу, відомого як спринт, зазвичай триває від одного до чотирьох тижнів. Протягом кожного спринту крос-функціональна команда зосереджується на доставці набору користувацьких історій, які представляють конкретні функціональності чи особливості програмного забезпечення. Щоденні стендап-зустрічі сприяють комунікації, сесії планування спринта визначають цілі, а перегляди спринта та ретроспективи забезпечують постійний зворотний зв'язок та адаптацію [22].

Хоча Scrum надає структуровану рамку, Agile-методології охоплюють ширший набір принципів. Ці принципи надають пріоритет ітеративному розвитку, постійному залученню клієнтів та адаптації до змін. Це сприяє більш реагуючому процесу розробки, дозволяючи командам швидко реагувати на змінні потреби користувачів та ринкові вимоги [23].

На завершення, Git, Scrum та Agile-методології дозволяють розробникам орієнтуватися в складнощах сучасної розробки програмного забезпечення. Надійні можливості контролю версій Git забезпечують стабільність та співпрацю, тоді як Scrum та принципи Agile сприяють гнучкості, постійній адаптації та швидкій доставці проектів.

У постійно змінному цифровому ландшафті забезпечення безпеки чутливих даних користувачів та підтримка безпеки веб-сайтів є ключовими питаннями. Ця складна мережа захисту утворена з трьох важливих елементів: HTTPS, SSL/TLS та проект OWASP. Розуміння їх взаємодії дозволяє власникам та розробникам веб-сайтів створювати безпечне середовище для онлайн-взаємодії.

HTTPS, або захищений протокол передачі гіпертексту, є основою безпечного веб-зв'язку. Він діє як безпечний канал, що шифрує передачу даних між веб-сервером та браузером користувача за допомогою протоколу Secure Sockets Layer/Transport Layer Security (SSL/TLS). Це шифрування робить дані нерозбірливими для будь-яких можливих сторонніх осіб, що перехоплюють мережу, захищаючи чутливу інформацію, таку як логіни та паролі, реквізити кредитних карт та особисті дані. Наявність HTTPS позначається замком у адресному рядку та "https://" перед URL, що свідчить про довірування з'єднанню [24].

Хоча HTTPS позначає безпечне з'єднання, шифрувальний механізм, що йому лежить в основі, забезпечується протоколами SSL/TLS. Ці протоколи встановлюють безпечний канал зв'язку, використовуючи два криптографічні ключі: публічний ключ та приватний ключ. Публічний ключ є загальнодоступним, тоді як приватний ключ залишається безпечно збереженим на сервері. Під час зв'язку дані шифруються за допомогою публічного ключа, і лише приватний ключ сервера може їх розшифрувати. Це забезпечує конфіденційність та цілісність даних, зменшуючи ризик порушення безпеки даних та атак типу "чоловік-посередник" [25].

Проте впровадження HTTPS та SSL/TLS само по собі не гарантує абсолютної безпеки. Проект OWASP виступає як ключовий ресурс у цьому контексті. OWASP - це некомерційна організація, яка присвячена покращенню безпеки веб-додатків. Вона надає комплексну структуру, OWASP Top 10, яка визначає десять найкритичніших ризиків безпеки веб-додатків. Ця структура дозволяє розробникам превентивно виявляти та вирішувати вразливості у своєму коді, зміцнюючи їхні програми проти загроз, таких як ін'єкційні атаки, порушення автентифікації та скриптові атаки [26].

На завершення слід відзначити, що HTTPS, SSL/TLS та OWASP утворюють міцний безпечний трикутник. HTTPS забезпечує безпечний зв'язок, SSL/TLS надає механізм шифрування, а OWASP надає розробникам знання для

побудови безпечних додатків. Впроваджуючи ці заходи, власники веб-сайтів можуть заслужити довіру користувачів та мінімізувати ризик порушення безпеки даних, захищаючи чутливу інформацію та забезпечуючи безпечне онлайн-середовище.

1.3. JavaScript як основна мова програмування у веб-розробці: значення та роль

JavaScript стоїть як фундаментальна опора в області динамічних веб-інтерфейсів. Його здатність маніпулювати (DOM), ієрархічною структурою, що представляє веб-сторінку, дозволяє робити зміни в реальному часі та взаємодію з користувачем, які перевищують обмеження статичного HTML. Це есе досліджує можливості JavaScript у створенні динамічних веб-інтерфейсів, досліджуючи його основні функціональні можливості та їх вплив на користувацький досвід [3].

Однією з переваг JavaScript є його здатність динамічно оновлювати вміст без повного перезавантаження сторінки. Традиційно, будь-яка зміна на веб-сторінці потребувала повного оновлення, що порушувало потік користувача. JavaScript уникнув цієї проблеми, дозволяючи цільові зміни у конкретні HTML-елементи. Припустімо ситуацію, де користувач вибирає продукт зі списку випадаючого меню. JavaScript може безперешкодно оновити відповідний розділ інформації про продукт, забезпечуючи плавний і ефективний досвід. Ця техніка, відома як Асинхронний JavaScript та XML (AJAX), революціонізувала розвиток веб-додатків, сприяючи більш інтерактивному користувацькому досвіду [3].

Крім того, JavaScript дозволяє створювати інтерактивні елементи, які підвищують залученість користувачів. Валідація форми є хорошим прикладом. Шляхом реалізації скриптів валідації, JavaScript може забезпечити відправку користувачем даних у правильному форматі, запобігаючи помилкам та оптимізуючи процеси відправки форми. Інтерактивні віджети, такі як слайдери,

каруселі та акордеони, також є продуктами майстерності JavaScript. Ці елементи не лише збагачують візуальну привабливість веб-сайту, але й надають користувачам інтуїтивні способи навігації та взаємодії з контентом.

Поза цими основними можливостями, JavaScript відкриває шлях для складних веб-додатків. Коли використовується разом з бібліотеками та фреймворками JavaScript, такими як React або Angular, розробники можуть створювати складні UI з многоразовими компонентами та можливостями зв'язування даних. Ці фреймворки пропонують структурований підхід до розробки веб-додатків, сприяючи збереженню коду та прискоренню циклів розробки.

Важливо визнати, що з великою потужністю приходить велика відповідальність. Неуклюжий код JavaScript може призвести до замедлення роботи та порушення користувацького досвіду. Кращі практики, такі як оптимізація коду та асинхронні техніки програмування, стають найважливішими в створенні ефективних та реагуючих веб-інтерфейсів.

JavaScript (JS) займає унікальне положення в ландшафті веб-розробки, вправно обслуговуючи як функціональності фронтенда, так і бекенда. Традиційно ці області вважалися різними, з HTML і CSS, що панують на фронтенді (клієнтській стороні), і серверними мовами програмування, такими як Java або Python, що оркеструють процеси на бекенді. Однак універсальність JavaScript розмиває ці межі, вводячи концепцію розробки повного стеку, де одна мова може керувати обома аспектами створення веб-додатків.

На фронтенді роль JavaScript добре відома. Як вже обговорювалося, його здатність маніпулювати DOM надає динамічного характеру статичним веб-сторінкам. Він полегшує оновлення динамічного контенту, інтерактивні елементи та безшовний користувацький досвід [3]. Фреймворки JavaScript, такі як React і Angular, додатково зміцнюють розробку фронтенду, пропонуючи модульність, зв'язування даних та оптимізовані процеси розробки. Ці

фреймворки, по суті, діють як готові набори інструментів, що дозволяють розробникам зосередитися на логіці застосунку, а не на тому, щоб винаходити колесо для загальних функціональностей.

Поява Node.js позначила зміну парадигми, вводячи еру JavaScript на бекенді. Node.js є середовищем виконання з відкритим вихідним кодом, яке виконує JavaScript-код за межами обмежень веб-браузера. Це дозволяє розробникам використовувати свої знання JavaScript для побудови масштабованих бекендових додатків. Архітектура Node.js на основі подій та неблокуюча модель зробили його особливо підходящим для реального часу застосунків та мікрослужб.

Проте відповідність JavaScript для розробки бекенда викликала дискусії. Хоча його легкість та широка підтримка спільноти неоспоримі переваги, існують обурення стосовно його однопотокового характеру та потенційної складності коду в проектах великого масштабу. Тут важливо визнати, що JavaScript не є універсальним засобом для розробки бекенда. Традиційні мови серверної частини можуть бути кращим вибором для конкретних сценаріїв, що вимагають високої продуктивності або складних обчислювальних завдань [3].

Узагальнюючи, JavaScript надає розробникам потужний набір інструментів, яким можна користуватися як на фронтенді, так і на бекенді. Рішення про те, де використовувати JavaScript, залежить від вимог проекту та експертизи команди. Для невеликих інтерактивних веб-додатків підхід з повним стеком JavaScript, використовуючи Node.js на бекенді, може бути дуже ефективним. Натомість для обчислювально інтенсивних завдань або великомасштабних корпоративних додатків встановлені мови серверної частини можуть бути більш обґрунтованим вибором.

У підсумку, універсальність JavaScript перетинає традиційний розрив між фронтендом і бекендом. Хоча його перевага на фронтенді залишається незаперечною, Node.js відкриває двері для можливостей розробки на бекенді.

По мірі того як веб-розробка продовжує розвиватися, здатність JavaScript адаптуватися та обслуговувати обидві сторони спектру розробки забезпечує його подальшу актуальність в постійно змінному технологічному ландшафті.

JavaScript (JS) закріпив своє місце як кутовий камінь веб-розробки. Його універсальність, велика екосистема бібліотек та фреймворків і постійна еволюція гарантують його подальшу домінантність у наступні роки.

Одним з провідних трендів є продовження зростання фреймворків JavaScript. React, Angular та Vue.js наразі панують, пропонуючи структуровані підходи до розробки фронтенду. Ми можемо очікувати подальшого розвитку цих фреймворків з фокусом на покращення продуктивності, поліпшення досвіду розробника та спрощення інтеграції передових веб-технологій. Крім того, поява нових фреймворків, пристосованих для конкретних ніш або функціональних можливостей, є виразною можливістю.

Вплив JavaScript, ймовірно, поширеться і поза браузером. Успіх Node.js на стороні сервера відкриває шлях до більш єдиної JavaScript-екосистеми. Ми можемо очікувати майбутнього, де розробка повного стеку зосереджуватиметься переважно навколо JavaScript, з Node.js в якості міцного фундаменту на бекенді поруч з встановленими фреймворками фронтенду. Це спростило б процеси розробки та, можливо, зменшило криву навчання для бажаючих розробників.

Майбутнє JavaScript, ймовірно, також буде пов'язане з еволюцією веб-технологій. Прогресивні веб-додатки (PWAs) - це чудовий приклад. PWAs розмивають межі між веб-додатками та мобільними додатками, пропонуючи досвід подібний до додатків всередині веб-браузера. JavaScript візьме на себе ключову роль у розробці цих веб-додатків нового покоління, забезпечуючи функції, такі як робота в автономному режимі та сповіщення про натискання.

Проте на горизонті також з'являються виклики. Постійно зростаюча складність кодових баз JavaScript вимагає більш суворого дотримання

найкращих практик та прийняття надійних методологій тестування. Крім того, питання безпеки залишаються постійною проблемою. По мірі розширення впливу JavaScript зростає й потенційна поверхня атак для зловмисників. Подолання цих ризиків потребуватиме постійних зусиль у сфері кращих практик безпеки та розвитку стандартів безпечного кодування.

Підсумовуючи, майбутнє JavaScript виглядає яскравим. Подальший розвиток фреймворків, його потенційна домінантність у повному стеку розробки та його співпраця з новими веб-технологіями, такими як PWAs, свідчать про продовження його панування в області веб-розробки. Проте вирішення викликів, пов'язаних із складністю коду та безпекою, буде найважливішим для забезпечення сталого зростання JavaScript та його здатності формувати майбутнє вебу.

1.4. Порівняльний аналіз JavaScript-бібліотек та фреймворків: вибір інструменту для розробки

JavaScript (JS) відіграє важливу роль у створенні динамічних та інтерактивних веб-сайтів. Проте, розробка складних веб-додатків з нуля може бути складною задачею. У таких випадках на допомогу приходять бібліотеки та фреймворки JavaScript, які надають готові функціональності та структуровані підходи для полегшення розробки. Хоча обидва ці інструменти підвищують ефективність розробки, вони відповідають різним потребам та пропонують різний рівень контролю [27].

Припустімо, що бібліотека - це як набір інструментів, що містить перевикористовувані компоненти. JavaScript-бібліотеки включають у себе набір написаних раніше модулів коду, які вирішують конкретні завдання. Такі бібліотеки, як jQuery, спрощують маніпулювання DOM та обробку подій, або Chart.js, спеціалізовані на генерації різних типів графіків. Розробники можуть використовувати ці бібліотеки у своїх проектах, не пишучи код з нуля.

Бібліотеки, як правило, мають менший обсяг коду порівняно з фреймворками, що робить їх ідеальними для вдосконалення конкретних аспектів додатків.

Фреймворк можна порівняти з попередньо розробленим архітектурним малюнком для будівництва будинку. JavaScript-фреймворки надають комплексну структуру для розробки веб-додатків. Вони встановлюють визначений спосіб організації коду, управління потоком даних та взаємодією з користувачем. Популярні фреймворки, такі як React або Angular, пропонують такі функції, як компонентна архітектура, прив'язка даних та можливості маршрутизації. Ці функції встановлюють певний стиль розробки, хоча часто потребують більше часу для вивчення, ніж бібліотеки. Проте фреймворки мають значні переваги в плані підтримки, масштабованості та організації коду для складних веб-додатків [27].

Вибір між бібліотеками та фреймворками залежить від потреб проекту та цілей розробки. Ось порівняльний розбір для допомоги в прийнятті рішення:

- Складність: Бібліотеки ідеально підходять для додавання конкретних функціональностей до існуючих кодових баз. Фреймворки, з іншого боку, краще підходять для побудови великих веб-додатків з нуля.
- Контроль: Бібліотеки пропонують високий рівень контролю над кодом. Фреймворки, з їхньою встановленою структурою, забезпечують меншу гнучкість, але гарантують послідовність та підтримку.
- Крива Навчання: Бібліотеки, як правило, легше вивчати та інтегрувати в існуючі проекти. Фреймворки вимагають більш значної інвестиції в освоєння їхнього специфічного синтаксису та шаблонів.

Отже, бібліотеки та фреймворки - це взаємодоповнюючі інструменти в арсеналі розробника JavaScript. Хоча бібліотеки пропонують покращення специфічних функціональностей, фреймворки надають структурований підхід для побудови складних веб-додатків. Розуміння їхніх сильних і слабких сторін

дозволяє розробникам приймати обґрунтовані рішення, що в кінцевому рахунку сприяє ефективній та підтримуваній веб-розробці.

Світ JavaScript (JS) має різноманітну колекцію бібліотек та фреймворків, кожен з яких призначений для конкретних потреб у розробці. Розуміння цих різних типів дозволяє розробникам вибирати найбільш підходящий інструмент для завдання. Тут ми детально розглянемо різні категорії та висвітлимо їх відмінні характеристики.

Категоризація за Функціональністю

Маніпулювання DOM та Покращення UI: Бібліотеки, такі як jQuery (універсальна), lodash (утиліти) та Bootstrap (CSS-фреймворк з компонентами JS), визнані лідерами у цій галузі. Вони надають безліч функцій для маніпулювання DOM, обробки подій та створення інтерактивних користувацьких інтерфейсів.

Обробка Даних та Візуалізація: Бібліотеки, такі як Moment.js (обробка дат/часу) та Chart.js (візуалізація даних), відповідають цим конкретним вимогам. Вони пропонують функціонал для розбору, форматування та візуалізації даних, збагачуючи веб-додатки цінними інсайтами.

Клієнтська Мережева Робота та AJAX: Бібліотеки, такі як Axios та Fetch API, спрощують асинхронну комунікацію між браузером та сервером. Вони дозволяють розробникам отримувати дані та динамічно оновлювати веб-сторінки без повного перезавантаження.

Фронтенд Фреймворки: Фреймворки, такі як React, Angular та Vue.js, представляють собою комплексний підхід до розробки фронтенду. Вони надають структурований спосіб створення користувацьких інтерфейсів, управління потоком даних та обробки взаємодії користувача. Ці фреймворки потребують більшого зусилля для вивчення, але сприяють організації коду, підтримці та масштабованості для складних веб-додатків [27].

Категоризація за Архітектурним Стилем

Фреймворки за MVC: Фреймворки, такі як Angular та Ember.js, дотримуються архітектурного шаблону MVC. Цей шаблон розділяє логіку застосунка (Модель), користувацький інтерфейс (Вид) та обробку взаємодії з користувачем (Контролер) на відокремлені шари, сприяючи модульності та перевикористанню.

Фреймворки на Основі Компонентів: Фреймворки, такі як React та Vue.js, пропагують компонентно-орієнтовану архітектуру. Додатки створюються за допомогою компонентів, які відповідають за конкретну функціональність або елемент користувацького інтерфейсу. Цей підхід сприяє швидкому розвитку та спрощує підтримку.

Фреймворки із Перевагою Використання Утиліт: Фреймворки, такі як Svelte та Marko, фокусуються на продуктивності та простоті використання. Вони пропонують більш легкий підхід порівняно з повноцінними фреймворками MVC, часто спираючись на розробника для управління станом та логікою застосунка [28].

Ландшафт JavaScript-бібліотек та фреймворків динамічний і постійно змінюється. Нові інструменти виникають для вирішення конкретних потреб, тоді як встановлені учасники отримують постійні оновлення та поліпшення. Слідкування за цими досягненнями дозволяє розробникам використовувати найефективніші інструменти для вирішення сучасних викликів веб-розробки.

Хоча всі відомі гравці в екосистемі JavaScript (JS), jQuery, Angular, React і Vue.js задовольняють різні потреби в розробці і пропонують різні рівні складності. У цій таблиці 1.1 наведено порівняльний огляд їх ключових характеристик.

Таблиця 1.1 – Порівняння ключових характеристик

Ознака	jQuery	Angular	React	Vue.js
Категорія	Бібліотека	Фреймворк (MVC)	Бібліотека (Заснована на компонентах)	Бібліотека (Заснована на компонентах)
Фокус	Маніпуляція DOM, Обробка подій, AJAX	Одно сторінкові додатки (SPA), Складні Інтерфейси	Створення пере використовуваних компонентів UI	Створення пере використовуваних компонентів UI
Крива Навчання	Відносно низька	Середня до Висока	Середня	Середня
Контроль	Високий	Нижчий (використовує структуру MVC)	Високий	Високий
Зв'язування даних	Ні	Двостороннє зв'язування даних	Одно направлений потік даних (може бути розширений)	Двостороннє зв'язування даних (опціонально)
Популярність	Висока	Висока	Дуже Висока	Висока
Спільнота та підтримка	Обширна	Обширна	Обширна	Обширна
Підходить для	Малі та середні інтерактивні елементи	Великі, складні SPA	Створення пере використовуваних компонентів UI, SPA	Створення пере використовуваних компонентів UI, SPA
Приклади	Вибір селекторів, анімації, валідація форм	Електронні комерційні додатки, платформи соціальних мереж	Новинні додатки, платформи для обміну фотографіями	Додатки типу "Список справ", системи управління контентом

Додатково розглянуто:

- jQuery: У той час як все ще популярний вибір для невеликих завдань маніпуляції DOM, його роль у сучасній веб-розробці зменшується. Сучасні фреймворки пропонують схожі функціональні можливості з більш структурованим підходом.
- Angular: зрілий і багатофункціональний фреймворк, Angular забезпечує певний стиль розробки і пропонує більш круту криву навчання. Він перевершує в створенні великих, корпоративних веб-додатків.
- React: архітектура на основі компонентів React сприяє повторному використанню та підтримуванню коду. Його орієнтація на

компоненти інтерфейсу робить його універсальним інструментом для створення інтерактивних елементів або складних SPA.

- Vue.js: Vue.js знаходить баланс між простотою і гнучкістю. Він пропонує зручний API, надаючи розширені функції, такі як двостороння прив'язка даних (необов'язково) [28].

РОЗДІЛ 2 ВИКОРИСТАННЯ REACT ДЛЯ РОЗРОБКИ САЙТУ ПУБЛІКАЦІЙ

2.1. React: основні принципи та можливості

React, бібліотека JavaScript для створення користувацьких інтерфейсів, стала домінуючою силою у веб-розробці з моменту свого створення в 2011 році. Спочатку створений Джорданом Вальке у Facebook для вирішення проблем із підтримкою швидко зростаючої кодової бази соціальної мережі, основні принципи роботи React з архітектурою на основі компонентів та віртуальними маніпуляціями DOM значно покращили продуктивність розробників та ремонтопридатність коду. Цей огляд досліджує історичний розвиток React, аналізуючи ключові віхи та технологічні досягнення, які сформували його поточний вигляд [29].

Рік 2013 відзначився переломним моментом з офіційним відкриттям коду React на GitHub. Це рішення сприяло створенню живої спільноти розробників, яка активно вносила свій внесок у зростання бібліотеки. Однією з ключових відмінностей від існуючих рішень був JSX, розширення синтаксису для JavaScript, яке дозволяло розробникам писати структури, схожі на HTML, прямо у своєму коді. Можна сказати, що JSX спрощував процес створення компонентів інтерфейсу користувача, можливо, зменшуючи навантаження на когнітивну функцію розробників.

Постійний розвиток React підтверджується серією значних оновлень. Введення класів у React 16.0 забезпечило більш структурований підхід до створення компонентів, тоді як наступне введення хуків у React 16.8 запропонувало функціональну альтернативу для управління станом компонентів та методами життєвого циклу. Цей перехід до хуків є помітним трендом, який може відображати вибір більш стислого та компонованого способу написання компонентів React [29].

Дивлячись поза основну бібліотеку, екосистема React процвітає. Інструменти, такі як Redux, виникли для вирішення складнощів управління станом у великомасштабних додатках, тоді як бібліотеки, такі як Next.js та Gatsby, спрощували створення веб-додатків з рендерингом на стороні сервера та статично генерованих веб-додатків, відповідно. Ця багата екосистема дозволяє розробникам використовувати сильні сторони React, вирішуючи потенційні недоліки.

Загалом, шлях React від внутрішнього рішення у Facebook до всезагального інструмента веб-розробки є свідченням його інноваційного підходу та сильної спільноти розробників. Постійний потік оновлень та процвітаюча екосистема, яка оточує React, свідчать про світле майбутнє для цієї бібліотеки. Проте, майбутні напрямки досліджень можуть досліджувати можливі компроміси, пов'язані з синтаксисом JSX, та довгострокову підтриму складних додатків React.

Успіх React можна віднести до декількох основних концепцій, які явно підвищили ефективність і ремонтопридатність веб-розробки. У цьому розділі розглядаються три основні принципи React: компоненти, віртуальний DOM і JSX.

React використовує архітектуру на базі компонентів, де складні користувацькі інтерфейси (UIs) розкладаються на повторно використовувані будівельні блоки. Кожен компонент інкапсулює свою власну логіку та представлення, сприяючи модульності та повторному використанню коду. Ця парадигма сприяє більш організованій кодовій базі, що потенційно зменшує навантаження на когнітивну функцію для розробників, які працюють над додатками великої масштабності [12].

Концепція компонентів узгоджується з принципом єдиної відповідальності (SRP) в програмній інженерії, який диктує, що компонент повинен мати єдину, чітко визначену відповідальність. Дотримання SRP у

React-компонентах може суттєво покращити ремонтопридатність та тестованість коду.

Одним із ключових нововведень React є віртуальний DOM, який є в пам'яті представленням реального DOM. Коли стан або властивості компонента змінюються, React ефективно обчислює мінімальний набір оновлень, необхідних у віртуальному DOM. Цей процес відмінює непотрібні маніпуляції реальним DOM, що може бути проблемою продуктивності у веб-додатках.

Віртуальний DOM надає значну перевагу продуктивності, особливо для додатків з частими оновленнями користувацького інтерфейсу. Хоча сама концепція не є унікальною для React, її реалізація демонструє покращення реагування та ефективності відтворення веб-додатків, побудованих з використанням цієї бібліотеки [12].

JSX, розширення синтаксису для JavaScript, дозволяє розробникам писати структури, схожі на HTML, прямо у своєму коді. Цей підхід сприяє більш природньому способу представлення компонентів інтерфейсу, потенційно зменшуючи навантаження на когнітивну функцію, пов'язану з управлінням окремими HTML- та JavaScript-файлами. Хоча JSX не є обов'язковим у React, він став широко прийнятим конвенцією у спільноті розробників React [12].

Проте використання JSX може внести потенційні недоліки. Деякі розробники вважають, що JSX може приховувати базовий код JavaScript, що може зробити його менш зрозумілим для тих, хто не знайомий з синтаксисом. Крім того, залежність від JSX може ускладнити можливості рендерингу на стороні сервера (SSR), що потрібно враховувати для певних архітектур веб-додатків [12].

Загалом, компоненти, віртуальний DOM та JSX формують основу React-розробки. Кожна концепція відіграє важливу роль у створенні та маніпулюванні ефективними користувацькими інтерфейсами. Хоча деякі аспекти, наприклад JSX, мають свої компроміси, основні принципи React демонструють очевидне

покращення того, як розробники веб-додатків будують та підтримують складні користувацькі інтерфейси.

Введення функціональних компонентів та хуків у React 16.8 відзначило значний зсув у бік функціональної парадигми програмування у екосистемі React [29].

Функціональні компоненти є основним концептом у функціональному підході до розробки React. На відміну від класових компонентів, які покладаються на методи життєвого циклу та управління станом у межах класу, функціональні компоненти є чистими функціями, які приймають властивості (props) як вхідні дані та повертають JSX, описуючи UI. Цей декларативний стиль сприяє більш стислому та зрозумілому способу написання компонентів React, що потенційно покращує підтримуваність коду.

Крім того, функціональні компоненти від природи мають переваги чистих функцій. Вони передбачувані, оскільки один і той самий вхід завжди виробляє однаковий вихід, і не мають побічних ефектів, що означає, що вони не прямо змінюють зовнішній стан. Ця характеристика може демонструвати спрощення в мисленні про поведінку компонента та сприяти юніт-тестуванню.

Проте, функціональні компоненти з складною логікою або вимогами до управління станом можуть стати важкими для читання та підтримки. У таких сценаріях використання компонентів більшого порядку або власних хуків може допомогти ізолювати складну логіку та покращити організацію коду.

Введення хуків у React 16.8 подальше зміцнення функціонального підходу. Хуки надають можливість "підключатися до" функцій React, таких як стан та методи життєвого циклу, з функціональних компонентів. Це усуває потребу в класових компонентах у багатьох ситуаціях, сприяючи більш послідовному та функціональному стилю коду по всьому додатку [29].

Хуки, подібно самим функціональним компонентам, сприяють більш композиційному підходу до створення UI. Можна створювати менші, повторно

використовувані функції для управління конкретними функціональностями всередині компонента, сприяючи повторному використанню коду та, можливо, зменшенню зайвості в кодовій базі.

Хоча функціональний підхід має значні переваги, важливо визнати можливі компроміси. Складне управління станом у великих додатках може стати важким у логічному відношенні з виключним використанням функціональних компонентів та хуків. У таких сценаріях використання бібліотеки управління станом, такої як Redux, може бути корисним [29].

Крім того, розробники, звиклі до об'єктно-орієнтованої парадигми у класових компонентах React, можуть потребувати періоду адаптації при переході до більш функціонального підходу.

На завершення, функціональний підхід у React, з акцентом на функціональні компоненти та хуки, пропонує привабливу альтернативу для створення користувацьких інтерфейсів. Декларативний стиль, вбудовані переваги чистих функцій та переваги композиційності сприяють чистому та підтримуваному коду. Однак потенційні складнощі, пов'язані з управлінням складним станом та крива навчання для деяких розробників, вимагають обдуманого підходу. Рішення про прийняття чисто функціонального підходу або використання комбінації функціональних та класових компонентів повинно бути прийняте на основі конкретних потреб та складності додатка.

Підхід до розробки на React: Функціональні компоненти проти Класових компонентів - Науковий аналіз

React пропонує два основних підходи для побудови UI-компонентів: функціональні компоненти та класові компоненти. Ця глава занурюється в науковий порівняльний аналіз цих підходів, аналізуючи їх переваги, недоліки та придатність для різних випадків використання.

Класові компоненти були традиційним підходом у розробці React з самого початку. Вони успадковують від класу React.Component та визначають

свій UI та поведінку, використовуючи методи, такі як render та методи життєвого циклу (наприклад, componentDidMount, componentDidUpdate). Цей об'єктно-орієнтований підхід дозволяє чітко розділити відповідальності та управління станом всередині компонента [30].

Переваги:

- **Підходить для Складної Логіки та Управління Станом**: Класові компоненти надають структурований спосіб обробки складної логіки та зміни стану за допомогою методів життєвого циклу. Це може бути вигідно для компонентів з витонченими потоками даних та взаємодіями.
- **Чітке Управління Життєвим Циклом**: Використання спеціальних методів життєвого циклу в класових компонентах явно визначає, коли мають відбуватися певні дії в життєвому циклі компонента. Це може покращити зрозумілість коду та його підтримуваність для розробників, що знайомі з концепціями об'єктно-орієнтованого програмування.

Недоліки:

- **Об'ємність**: Класові компоненти можуть стати об'ємними, особливо для простіших компонентів з мінімальною логікою. Визначення класу, методу render і, можливо, кількох методів життєвого циклу може призвести до більшого обсягу коду порівняно з функціональними компонентами.
- **Складність Мислення**: У компонентів з обширними методами класу та гаками життєвого циклу може виникнути складність при розумінні загальної поведінки компонента. Це може ускладнити відлагодження та підтримку коду.

Функціональні компоненти є чистими функціями JavaScript, які приймають властивості (props) як вхідні дані та повертають JSX, що описує UI.

Цей декларативний стиль сприяє більш стислому та зрозумілому визначенню компонентів, особливо для простіших UI. Введення гаків у React 16.8 додатково зміцнило функціональні компоненти, надаючи механізм "підключення" до функцій React, таких як стан та методи життєвого циклу [30].

Переваги:

- **Читабельність та Підтримуваність**: Функціональні компоненти з гаками, як правило, є більш стислими та легкими для читання порівняно з класовими компонентами. Це може бути особливо корисним для менших і менш складних компонентів.
- **Композиційність**: Функціональні компоненти, за своєю природою, сприяють більш композиційному підходу до створення UI. Можна створювати менші, повторно використовувані функції для управління конкретними функціональностями, що сприяє повторному використанню коду та, можливо, зменшенню зайвості в кодовій базі.
- **Чисті Функції**: Функціональні компоненти (без побічних ефектів) з природи користуються перевагами чистих функцій. Вони передбачувані та легкі для тестування завдяки відсутності неочікуваної поведінки.

Недоліки:

- **Складне Управління Станом**: Хоча гаки надають механізми для управління станом, обробка складної логіки стану всередині функціональних компонентів може стати незручною. У таких сценаріях використання бібліотеки управління станом, такої як Redux, може бути більш відповідним.
- **Крива Навчання**: Розробники, звиклі до об'єктно-орієнтованої парадигми у класових компонентах React, можуть потребувати

періоду адаптації при переході до функціонального підходу з гаками.

Рішення про використання функціональних компонентів чи класових компонентів повинно базуватися на конкретних потребах компонента та загальній архітектурі додатка [30].

- **Функціональні компоненти** з гаками є переконливим вибором для простіших UI, повторно використовуваних компонентів UI та ситуацій, де потрібна чітка роздільність відповідальностей. Їх стислий синтаксис та вбудовані переваги чистих функцій сприяють читабельності та підтримуваності коду.
- **Класові компоненти** залишаються актуальними для компонентів з складною логікою, витонченими вимогами до управління станом чи для розробників, які віддають перевагу структурованому об'єктно-орієнтованому підходу. Їх методи життєвого циклу надають чіткий контроль над поведінкою компонента протягом всього його життєвого циклу.

Як функціональні, так і класові компоненти мають своє місце в наборі інструментів розробника React. Розуміння їх переваг і недоліків дозволяє приймати обґрунтовані рішення при створенні UI-компонентів. Майбутнє розробки React, здається, віддає перевагу більш функціональному підходу з гаками, але класові компоненти ймовірно залишаться цінним інструментом для конкретних випадків використання.

React Hooks, представлені у версії 16.8, представляють значну інновацію в екосистемі React. Вони забезпечують потужний механізм використання стану та інших можливостей React з функціональних компонентів, що потенційно призводить до більш стислого та композиційного підходу до побудови інтерфейсів користувача. Ця глава заглиблюється у основні концепції гаків, досліджуючи їх функціональність та потенційний вплив на розробку React.

До появи гаків управління станом у функціональних компонентах було незручним. Розробники часто вдавалися до методів, таких як "prop drilling" (передача даних через ієрархію компонентів), що може призводити до складного та менш підтримуваного коду. Гаки вирішили це обмеження, представивши гак useState. Цей гак дозволяє функціональним компонентам "прикріплюватися" до можливостей управління станом React, що дозволяє створювати та змінювати стан компонента безпосередньо всередині самого функціонального компонента.

Функціональність гаків розширюється поза управлінням станом. Додаткові базові гаки, такі як useEffect, надають доступ до методів життєвого циклу, таких як componentDidMount та componentDidUpdate, всередині функціональних компонентів. Це надає розробникам можливість виконувати побічні ефекти, підписки та отримання даних безпосередньо всередині функціональних компонентів, подальше зменшення залежності від класових компонентів.

Використання гаків, спільно з функціональними компонентами, має кілька потенційних переваг:

- **Покращена Читабельність та Підтримуваність**: Функціональні компоненти з гаками, як правило, є більш стислими та легшими для читання порівняно з класовими компонентами. Це може бути особливо корисним для менших та менш складних компонентів.
- **Підвищена Композиційність**: Гаки сприяють більш композиційному підходу до побудови інтерфейсів. За допомогою гаків можна створювати менші, повторно використовувані функції для управління конкретними функціональностями, що сприяє повторному використанню коду та зменшенню залежності.
- **Чисті Функції**: Функціональні компоненти з гаками (без побічних ефектів) з природи користуються перевагами чистих функцій. Вони

передбачувані та легкі для тестування завдяки відсутності неочікуваної поведінки.

Хоча гаки мають значні переваги, важливо враховувати потенційні обмеження:

- **Крива Навчання**: Розробники, звиклі до об'єктно-орієнтованої парадигми у класових компонентах React, можуть потребувати періоду адаптації при переході до функціонального підходу з гаками.
- **Складне Управління Станом**: Хоча гаки надають механізми для управління станом, обробка складної логіки стану у дуже великих додатках може стати незручною. У таких сценаріях використання бібліотеки управління станом, такої як Redux, може бути більш відповідним.

Поява гаків означає значний зміщення в бік більш функціонального підходу до розробки React. Хоча класові компоненти імовірно не зникнуть повністю, гаки пропонують переконливу альтернативу для побудови інтерфейсів. Їх стислий синтаксис, композиційність та відповідність принципам чистих функцій позиціонують їх як потужний інструмент у наборі інструментів розробника React.

2.2. Axios: HTTP-запити та робота з API

У постійно змінному пейзажі бібліотек JavaScript Axios з'явився як популярний вибір для здійснення HTTP-запитів у веб-додатках. Ця розділ глибше занурюється у основні функціональності Axios, аналізуючи його призначення та переваги порівняно зі вбудованими в браузер API, такими як Fetch [31].

Хоча API Fetch, представлений у ES6, забезпечує стандартизований спосіб здійснення HTTP-запитів, Axios пропонує більш зручну та функціонально багатшу альтернативу. Axios спрощує процес здійснення HTTP-

запитів, пропонуючи чистий синтаксис, автоматичне розбір JSON та вбудовані механізми обробки помилок.

Axios забезпечує лаконічний та інтуїтивно зрозумілий API для здійснення різних HTTP-запитів, включаючи GET, POST, PUT, DELETE та інші. Він дозволяє розробникам вказувати параметри запиту, заголовки та дані простим та зрозумілим способом. Крім того, Axios автоматично розбирає відповіді JSON, усуваючи потребу в ручному розборі за допомогою API Fetch [31].

Значною перевагою Axios є його потужні можливості обробки помилок. Він забезпечує послідовний спосіб обробки помилок під час обробки запитів та відповідей, покращуючи читабельність та підтримуваність коду. Крім того, Axios пропонує концепцію інтерсепторів, які є функціями, які можуть використовуватися для перехоплення запитів та відповідей перед тим, як їх обробляє основний додаток. Ця функціональність дозволяє розробникам реалізовувати функції, такі як глобальна аутентифікація запиту або ведення журналу помилок по всьому додатку.

Axios використовує Обіцянки для асинхронного спілкування, гармонійно поєднуючись з сучасними практиками JavaScript. Це дозволяє розробникам писати чистий та зрозумілий код при роботі з асинхронними операціями, такими як HTTP-запити.

Однією з ключових переваг Axios є його можливість працювати як у браузерному, так і в середовищі Node.js. Це робить його універсальним інструментом для побудови веб-додатків та додатків на стороні сервера, які потребують спілкування з API.

Хоча Axios пропонує значні переваги, важливо враховувати потенційні компроміси. На відміну від API Fetch, яке є вбудованою функцією браузера, Axios вводить додаткову залежність в проект. Однак переваги чистого API, автоматичного розбору JSON та потужної обробки помилок часто переважають цей незначний недолік.

Axios встановив себе як цінний інструмент для сучасної розробки вебу. Його зручний API, автоматичний розбір JSON, механізми обробки помилок, підхід на основі обіцянок та можливість працювати як у браузерному, так і в середовищі Node.js роблять його привабливим вибором для розробників, які шукають більш зручний та ефективний спосіб здійснення HTTP-запитів. Хоча API Fetch залишається життєздатною опцією, Axios пропонує функціонально багатший альтернативний варіант, який може спростити робочі процеси та покращити підтримуваність коду.

Axios, популярна бібліотека клієнта HTTP для JavaScript, пропонує потужний та зручний спосіб взаємодії з API у веб-додатках. Цей розділ досліджує практичні аспекти використання Axios для здійснення HTTP-запитів, наводячи основні функціональності та надаючи приклади коду для типових випадків використання.

Значною перевагою Axios є автоматичний розбір JSON. Дані відповіді автоматично розбираються в об'єкт JavaScript, усуваючи потребу у ручному розборі за допомогою API Fetch. Це спрощує доступ до даних та їх обробку в додатку.

Axios забезпечує структурований підхід до обробки помилок. Блок catch у ланцюжку обіцянок дозволяє розробникам гідно обробляти помилки та запобігати неочікуваній поведінці в додатку.

Axios пропонує потужну концепцію, яку називають інтерсепторами. Це функції, які можуть бути використані для перехоплення запитів та відповідей перед тим, як їх обробить основний додаток. Ця функціональність дозволяє для централізованої логіки, такої як додавання аутентифікаційних токенів до всіх запитів або ведення журналу викликів API.

Використовуючи можливості Axios, розробники можуть оптимізувати процес здійснення HTTP-запитів у своїх веб-додатках. Його зручний API, автоматичний розбір JSON, надійна обробка помилок та підхід на основі

обіцянок роблять його привабливим вибором для сучасної веб-розробки. Хоча API Fetch залишається життєздатною опцією, Axios пропонує багатофункціональну альтернативу, яка може спростити робочі процеси, покращити підтримуваність коду та покращити загальний досвід розробника.

Архітектура на основі компонентів та декларативний характер React роблять його потужним інструментом для створення інтерфейсів користувача. Однак додатки часто потребують даних з API для заповнення та оновлення їх елементів інтерфейсу користувача. Цей розділ досліджує інтеграцію Axios, популярної бібліотеки клієнта HTTP, з React, наводячи процес отримання даних з API та управління їх станом у компонентах React.

Зазвичай компоненти React управляють своїм станом, використовуючи хук useState. Отримані дані з API зберігаються у стані компонента за допомогою setData. Цей стан можна використовувати для умовного відображення елементів інтерфейсу користувача у вказаному вказанні компонента. Крім того, обробка помилок може бути реалізована за допомогою хука useState та блоку catch у запиті Axios для відображення відповідних повідомлень про помилки, якщо отримання даних не вдалося.

Хук useEffect є ключовим концептом для отримання даних у компонентах React. Цей хук дозволяє розробникам виконувати побічні ефекти, такі як виклики API, після рендерингу компонента. У вищенаведеному прикладі хук useEffect використовується для отримання даних під час монтування компонента ([] як масив залежностей). Це забезпечує, що дані отримуються лише один раз під час першого рендерингу компонента.

Базова інтеграція Axios з React дозволяє просте отримання даних. Однак для більш складних сценаріїв виникають додаткові умови. Компоненти можуть відображати індикатор завантаження під час отримання даних для покращення досвіду користувача. Пагінацію можна реалізувати для роботи з великими наборами даних, отриманими з API. Бібліотеки, такі як Redux, можна

використовувати для складного управління станом, коли потрібно працювати з декількома компонентами, які залежать від одних і тих самих отриманих даних.

Переваги Інтеграції Axios

Інтеграція Axios з React пропонує кілька переваг для веб-розробки:

- **Спрощене Отримання Даних**: Axios надає лаконічний та зручний API для виконання HTTP-запитів, спрощуючи процес отримання даних з API у компонентах React.
- **Покращена Читабельність Коду**: Відокремлення логіки отримання даних від логіки інтерфейсу за допомогою Axios та хука useEffect може покращити читабельність та підтримуваність коду.
- **Надійна Обробка Помилок**: Механізми обробки помилок Axios дозволяють гідно обробляти невдачі запитів API, покращуючи стійкість додатка.

Комбінація Axios та React - це потужний підхід для створення сучасних веб-додатків. Axios пропонує спрощений спосіб взаємодії з API, а React забезпечує декларативну та компонентну архітектуру для побудови динамічних інтерфейсів. Використовуючи цю інтеграцію, розробники можуть створювати ефективні та підтримувані додатки, які ефективно отримують та використовують дані зовнішніх джерел.

РОЗДІЛ 3 ПРАКТИЧНЕ ВПРОВАДЖЕННЯ JAVASCRIPT-БІБЛІОТЕК ТА ФРЕЙМВОРКІВ: РОЗРОБКА ІНТЕРАКТИВНОГО ІНТЕРФЕЙСУ ВЕБ-САЙТУ

3.1 Розробка інтерактивного інтерфейсу веб-сайту за допомогою React

У даному розділі буде детально розглянуто процес створення компонентів React для веб-сайту блогу. React, як одна з найпопулярніших JavaScript-бібліотек для розробки інтерфейсів, надає широкий спектр можливостей для створення динамічних та інтерактивних веб-додатків.

Переваги використання React для створення блогу

Однією з ключових переваг React є його компонентний підхід. React дозволяє розробляти компоненти – незалежні, перевикористовувані блоки, які слугують будівельними елементами інтерфейсу. Це суттєво спрощує розробку та підтримку коду, адже складні інтерфейси розбиваються на простіші, лаконічні частини.

У контексті розробки блогу React надає наступні переваги:

- **Логічна ієрархія компонентів**: React дозволяє організувати компоненти в чітку ієрархію, що значно спрощує їх подальше керування та підтримку. Наприклад, можна створити окремі компоненти для заголовка блогу, окремі для публікацій та окремі для коментарів.
- **Взаємодія компонентів**: React надає зручні механізми взаємодії між компонентами. За допомогою властивостей (props) дані передаються по ієрархії компонентів, роблячи їх більш конфігурованими та універсальними. Крім того, зворотні виклики (callbacks) дозволяють отримувати оновлені дані від дочірніх компонентів, забезпечуючи ефективне керування станом програми.

- **Створення та експорт компонентів**: React дає можливість створювати компоненти, які можна легко експортувати та використовувати повторно. Це робить розробку більш швидкою та ефективною, адже з'являється можливість будувати бібліотеки компонентів для власних проектів або спільноти.

React, завдяки своїм можливостям, є чудовим інструментом для розробки веб-сайтів блогів. Компонентний підхід, зручні механізми взаємодії та можливість повторного використання компонентів роблять процес розробки динамічних та інтерактивних блогів значно простішим та ефективнішим.

Функціональність компонента "PopularPost"

```
// Компонент PopularPost приймає деструктуризовані пропси: id, title, date, user,
tags
export const PopularPost = ({ id, title, date, user, tags }) => {
    // Форматуємо дату для відображення
    const formattedDate = formatDate(date);
    // Повертаємо JSX для відображення популярного поста
    return (
        // Посилання на конкретний пост за його id
        <Link to={`/falts/post/${id}`}>
            <div className='popular-post'>
                <div className='info'>
                    <div className='author-info'>
                        <div className='author-block'>
    {/* Відображаємо зображення автора, його повне ім'я та відформатовану дату */}
                            <img src={user.image} alt={user.fullName} />
                            <p className='author'>{user.fullName}</p>
                            <p className='dot'>&#x2022;</p>
                            <p className='date'>{formattedDate}</p>
                        </div>
                        {/* Відображаємо іконку для збереження поста */}
                        <div className='save-icon'>
                            <SelectedIcon />
                        </div>
                    </div>
                    <div className='text-info'>
                        {/* Відображаємо заголовок поста */}
                        <h3 className='title'>{title}</h3>
                    </div>
                    <div className='actions'>
                        <div className='tags'>
                {/* Перебираємо та відображаємо всі теги, прив'язані до поста */}
```

```
                        {tags.map(tag => (
                            <div key={tag.id} className='tag'>
                                <p>{tag.name}</p>
                            </div>
                        ))}
                    </div>
                </div>
            </div>
        </Link>
    );
};
```

Рис. 1.1 Популярний пост

Компонент "PopularPost" (див. Додаток А1) відповідає за рендеринг інформації про популярні публікації блогу. Він приймає наступні параметри:

- postId: Ідентифікатор публікації
- title: Заголовок публікації
- date: Дата публікації
- author: Інформація про автора публікації
- tags: Список тегів, що використовуються у цій публікації

Сам компонент складається з наступних елементів:

- Блок з інформацією про автора та датою публікації: Цей блок містить ім'я автора та дату публікації.
- Список тегів: Цей список відображає всі теги, що використовуються у цій публікації.
- Іконка "SelectedIcon": Ця іконка дозволяє користувачеві зберегти публікацію.

Використання компонента "PopularPost"

```
// Імпортуємо компоненти Post та PopularPost з файлу Post.js у директорії
components/Post
import { Post, PopularPost } from '../../components/Post/Post';
/* . . . */
// Використовуємо метод map для перебору масиву data та відображення кожного поста
{data.map((post, index) => (
// Використовуємо React.Fragment для обгортки кожного поста та уникнення зайвих div
```

```
    <React.Fragment key={post.id}>
        <div>
         {/* Відображаємо компонент PopularPost з передачею необхідних пропсів */}
            <PopularPost
                user={users.find(user => user.id === post.user_id)}  // Знаходимо
користувача за його id
                title={post.title}  // Передаємо заголовок поста
                date={post.date}  // Передаємо дату поста
                id={post.id}  // Передаємо id поста
                tags={post.tags}  // Передаємо теги поста
            />
        </div>
        {/* Відображаємо розділювач між постами, окрім останнього поста */}
        {index !== data.length - 1 && <div className='divider'></div>}
    </React.Fragment>
))}
```

Рис. 1.2 Відображення списку популярних постів

У другій частині коду компонент "PopularPost" імпортується у файл, де планується відображати список популярних публікацій. Далі цей компонент використовується для рендерингу кожної публікації зі списку "data".

Під час рендерингу списку популярних публікацій код проходить через кожен елемент списку "data" та передає відповідні дані в компонент "PopularPost". Ці дані включають інформацію про автора, заголовок, дату, ідентифікатор публікації та список тегів. Компонент "PopularPost" потім використовує ці дані для рендерингу інформації про публікацію.

Переваги використання компонента "PopularPost"

Використання компонента "PopularPost" для відображення популярних публікацій блогу має ряд переваг:

- **Чистий та організований код**: Завдяки використанню компонента код стає більш чітким та структурованим, що полегшує його читання та розуміння.
- **Повторне використання**: Компонент "PopularPost" можна використовувати повторно для відображення будь-якої публікації блогу, що робить код більш гнучким та економним.

- **Легкість підтримки**: У разі виникнення необхідності змінити спосіб відображення популярних публікацій, це можна зробити, змінивши лише код компонента "PopularPost", не впливаючи на інші частини коду.

Компонент "PopularPost" є чудовим прикладом того, як React можна використовувати для створення чіткого, організованого та повторно використовуваного коду. Використання цього компонента значно спрощує процес відображення популярних публікацій блогу та робить код більш гнучким та економним.

Структура навігаційного меню

Навігаційне меню (див. Додаток А2) представлене списком <li>, кожен з яких має клас "filter" та відповідний текстовий вміст. Клас "active" використовується для позначення активного елемента навігації, який обирається користувачем.

```
// Функція handleActiveFilterPost обробляє подію натискання на елементи навігації
фільтрів
function handleActiveFilterPost(event) {
    // Знаходимо всі елементи з класом 'filter'
    const navItems = document.querySelectorAll('.filter');
    // Видаляємо клас 'active' з усіх елементів навігації
    navItems.forEach(item => {
        item.classList.remove('active');
    });
    // Додаємо клас 'active' до натиснутого елемента
    event.target.classList.add('active');
}
/* . . . */
// Відображаємо навігаційне меню з фільтрами
<nav className='filter-posts'>
    {/* Створюємо елементи списку, кожен з яких є фільтром */}
    <li className='filter active' onClick={handleActiveFilterPost}>Моя Стрічка</li>
    <li className='filter' onClick={handleActiveFilterPost}>Слідкую</li>
    <li className='filter' onClick={handleActiveFilterPost}>Популярне</li>
    <li className='filter' onClick={handleActiveFilterPost}>Нове</li>
</nav>
```

Рис. 1.3 Навігаційне меню

Функція handleActiveFilterPost відповідає за динамічне оновлення стилізації елементів навігаційного меню в залежності від вибору користувача. Її робота ґрунтується на наступних принципах:

1. **Визначення події кліку**: Функція приймає параметр event, який представляє подію кліка на елементі навігації.
2. **Видалення класу "active" з усіх елементів**: За допомогою event.target.classList.remove("active") клас "active" видаляється з всіх елементів навігаційного меню.
3. **Додавання класу "active" до обраного елемента**: За допомогою event.target.classList.add("active") клас "active" додається до елемента, на який було натиснуто.

Переваги даного підходу

Цей підхід до управління класами має ряд переваг:

- **Динамічна стилізація**: Стилізація елементів навігаційного меню оновлюється динамічно, відображаючи поточний вибір користувача.
- **Простота реалізації**: Код є простим та лаконічним, що робить його легким для розуміння та модифікації.
- **Відсутність необхідності зберігання стану**: Цей підхід не потребує зберігання стану в компоненті, що може бути корисним у випадках, коли ви працюєте з невеликими взаємодіючими елементами.

Реалізація навігаційного меню для фільтрації публікацій блогу, представлена у даному коді, є простим та ефективним рішенням. Завдяки використанню функції handleActiveFilterPost та динамічного управління класами, код стає чітким, лаконічним та легко модифікованим.

У цій роботі для стилізації веб-сайту було використано SCSS (Sassy CSS) – розширену версію CSS. Вибір SCSS ґрунтується на його перевагах перед звичайним CSS, які включають розширені можливості та спрощений синтаксис.

Переваги використання SCSS

1. **Використання змінних кольорів**

Однією з ключових переваг SCSS є можливість створювати змінні для зберігання значень кольорів. Це значно спрощує процес розробки та модифікації дизайну, адже для зміни кольорової гами веб-сайту достатньо лише змінити значення змінних.

У цій роботі були створені змінні $background-color-primary та $background-color-secondary, які зберігають значення кольорів для фону сайту. Завдяки цьому зміна кольорової гами стає надзвичайно швидкою та простою, адже не потребує редагування великої кількості CSS-коду.

```
// Основний фон для контенту сторінки
$background-background-primary: #18181B;
// Вторинний фон для секцій або елементів
$background-background-secondary: #27272A;
// Акцентний колір для обводки елементів
$stroke-stroke-accent: #FF8906;
// Основний колір для обводки елементів
$stroke-stroke-based: #27272A;
// Приглушений колір для обводки
$stroke-stroke-subdued: #27272A;
// Основний колір для іконок
$icon-icon-base: #FFFFFE;
// Приглушений колір для іконок
$icon-icon-subdued: #A1A1AA;
// Акцентний колір для іконок
$icon-icon-accent: #FF8906;
```

Рис. 1.4 Змінні для налаштування оформлення

2. **Стандартизація стилів тексту**

За допомогою SCSS та міксіна heading-1 вдалося стандартизувати стилі текстових елементів на веб-сайті. Це значно покращило візуальну єдність інтерфейсу та спростило його підтримку.

```
// Міксин для стилізації заголовка першого рівня
@mixin heading-1 {
    font-family: 'Mulish'; // Встановлює шрифт 'Mulish' для тексту
    font-size: 36px; // Встановлює розмір шрифту 36 пікселів
    font-weight: 800; // Встановлює товщину шрифту на 800 (дуже жирний шрифт)
    line-height: 40px; // Встановлює висоту рядка 40 пікселів для забезпечення
інтервалу між рядками
    letter-spacing: 0; // Встановлює інтервал між буквами на 0 (немає додаткового
простору між літерами)
}
```

Рис. 1.5 Міксин для стилізації заголовка першого рівня

3. Створення динамічних стилів

SCSS дозволяє створювати динамічні стилі, які адаптуються до різних умов. Це робить інтерфейс більш гнучким та зручним у користуванні.

Наприклад, у цій роботі клас .text-info використовує динамічні стилі для заголовків та тексту, що дає можливість їх легко використовувати у різних частинах веб-сайту.

```
.text-info {
    display: flex; // Встановлює гнучкий контейнер
    flex-direction: column; // Встановлює напрямок розташування елементів по
вертикалі (колонка)
    gap: 8px; // Встановлює відстань між елементами в колонці

    .title {
        @include heading-2; // Використовує міксин heading-2 для стилізації
заголовка
        color: $text-text-base; // Встановлює колір тексту для заголовка
    }

    .text {
        @include body-2; // Використовує міксин body-2 для стилізації основного
тексту
        color: $text-text-base; // Встановлює колір тексту для основного тексту
    }
}
```

Рис. 1.6 Стилізація блоку з текстовою інформацією

Використання SCSS у цій роботі є обґрунтованим та дозволяє покращити якість та продуктивність розробки веб-сайту. Його розширені можливості в

порівнянні зі звичайним CSS роблять SCSS цінним інструментом для сучасних веб-розробників.

3.2 Взаємодія з API за допомогою Axios

У ході розробки програмного забезпечення, особливо веб-додатків та API, часто виникає потреба у створенні тестових даних для перевірки функціональності та налагодження. Mokky - це простий сервіс, який полегшує цю задачу, надаючи зручний інтерфейс для створення тестових API, генерування даних користувачів та маніпулювання ними за допомогою готових REST API-методів.

API (Application Programming Interface) - це інтерфейс програмування застосунків, який визначає набір правил та специфікацій, що дозволяють різним програмним компонентам взаємодіяти один з одним. API надає абстракцію, яка приховує внутрішню реалізацію компонента, дозволяючи іншим програмам використовувати його функціональність без необхідності знати про його складні деталі.

REST API (Representational State Transfer Application Programming Interface) - це тип архітектури API, що ґрунтується на принципах передачі стану подання (REST). REST API використовують стандартні HTTP-методи (GET, POST, PUT, DELETE) для виконання операцій над ресурсами, таких як створення, читання, оновлення та видалення даних. REST API відомі своєю простотою, зрозумілістю та масштабованістю, що робить їх популярним вибором для розробки веб-сервісів.

Переваги використання Mokky

Mokky пропонує ряд переваг для розробників, які потребують тестових даних та API:

- **Простота використання**: Mokky має зручний веб-інтерфейс, який не потребує знань програмування для створення тестових API та генерування даних користувачів.
- **Гнучкість**: Mokky дозволяє генерувати різні типи даних користувачів, включаючи імена, адреси електронної пошти, паролі та іншу інформацію.
- **Готові REST API методи**: Mokky надає готові REST API методи для виконання операцій над даними користувачів, таких як створення, читання, оновлення та видалення.
- **Легкість інтеграції**: Mokky можна легко інтегрувати з іншими інструментами та платформами розробки.

Mokky - це цінний інструмент для розробників, які потребують тестових даних та API для своїх пет-проектів. Завдяки простоті використання, гнучкості та готовим REST API методам Mokky може значно полегшити процес тестування та налагодження програмного забезпечення.

В рамках даного дослідження розроблено механізм автоматизованого отримання даних з онлайн сервісу API, що використовує хук useEffect та бібліотеку Axios. Цей механізм гарантує, що дані завантажуються лише один раз при першому рендері компонента, оптимізуючи продуктивність та запобігаючи зайвим зверненням до API.

Для реалізації даного механізму використовується хук useEffect, який приймає два аргументи: callback-функцію та масив залежностей. Callback-функція виконується лише при зміні залежностей, що у даному випадку порожній масив, гарантуючи одноразове виконання при першому рендері.

Внутрішньо callback-функція fetchData асинхронно здійснює GET-запити до трьох ендпоінтів API (див. Додаток А3):

- **https://04cb5470549a62ec.mokky.dev/posts** - для отримання даних про публікації;

- **https://04cb5470549a62ec.mokky.dev/users** - для отримання даних про користувачів;
- **https://04cb5470549a62ec.mokky.dev/tags** - для отримання даних про теги.

Отримані дані обробляються та трансформуються наступним чином:

- **Дані про теги**: До кожного об'єкта тега додається нова властивість selected, яка вказує, чи був даний тег раніше збережений у локальному сховищі localStorage. Значення цієї властивості визначається на основі значення ключа tag_${tag.id} у localStorage.
- **Інші дані**: Отримані дані про користувачів та публікації зберігаються у відповідних стейтах компонету за допомогою функцій setUsers, setData, setInitialData та setTags, роблячи їх доступними для подальшого використання.

У разі виникнення помилки при отриманні даних з API, відповідне повідомлення про помилку виводиться у консоль розробника.

```
useEffect(() => {
    // Асинхронна функція для отримання даних
    const fetchData = async () => {
        try {
         // Виконуємо паралельні запити для отримання постів, користувачів та тегів
            const [postsResponse, usersResponse, tagsResponse] = await
Promise.all([
                axios.get('https://04cb5470549a62ec.mokky.dev/posts'),
                axios.get('https://04cb5470549a62ec.mokky.dev/users'),
                axios.get('https://04cb5470549a62ec.mokky.dev/tags')
            ]);

 // Оновлюємо теги, додаючи властивість 'selected' на основі значень з localStorage
            const updatedTags = tagsResponse.data.map((tag) => ({
                ...tag,
                selected: localStorage.getItem(`tag_${tag.id}`) === 'true',
            }));

            // Оновлюємо стан з отриманими даними
            setUsers(usersResponse.data);        // Зберігаємо користувачів у стані
            setData(postsResponse.data);         // Зберігаємо пости у стані
```

```
            setInitialData(postsResponse.data);  // Зберігаємо початкові дані
постів у стані
            setTags(updatedTags);                // Зберігаємо оновлені теги у
стані
        } catch (error) {
            // Логування помилки у випадку невдалого запиту
            console.error('Error fetching data:', error);
        }
    };

    // Викликаємо функцію для отримання даних
    fetchData();
}, []); // Залежність масиву порожня, тому useEffect виконується тільки один раз
при першому рендері компонента
```

Рис. 2.1 Завантаження даних з API та ініціалізація стану

Завдяки даній реалізації, процес отримання даних з API стає автоматизованим та ефективним, гарантуючи завантаження даних лише при першому рендері компонента, а також обробку та збереження даних у зручному форматі для подальшого використання.

Реалізація динамічного пошуку постів за допомогою хука useEffect

В рамках даної роботи розроблено механізм динамічного пошуку постів, який ґрунтується на використанні хука useEffect та функції handleSearch. Цей механізм дозволяє користувачам знаходити пости за ключовими словами в їх заголовках.

Функціональність пошуку реалізована наступним чином:

- **Використання хука useEffect**: Хук useEffect активується при зміні значення змінної searchTerm, яка відповідає за ключове слово для пошуку.
- **Перевірка порожнього значення searchTerm**: Якщо значення searchTerm порожнє, то хук useEffect встановлює початковий набір даних постів у стан data.
- **Пошук постів**: У випадку, коли searchTerm не порожній рядок, хук useEffect викликає функцію fetchData, яка відправляє GET-запит на сервер. У цьому запиті використовується параметр

title=*${searchTerm}, який дозволяє знаходити пости, заголовки яких містять вказане ключове слово.

- **Зберігання результатів пошуку**: Отримані з сервера результати пошуку зберігаються у стані data, який відповідає за список постів, що відповідають введеному ключовому слову.
- **Функція handleSearch**: Ця функція активується при зміні значення поля пошуку. Вона оновлює значення searchTerm, запускаючи тим самим ефект useEffect для виконання пошуку з урахуванням нового ключового слова.

Завдяки даній реалізації, пошук постів стає динамічним та ефективним. Користувачі можуть вводити ключові слова у поле пошуку, а сторінка автоматично оновлюється, відображаючи лише ті пости, які відповідають введеному запиту.

```
useEffect(() => {
    // Якщо пошуковий запит порожній, відновлюємо початкові дані
    if (searchTerm === '') {
        setData(initialData);
    } else {
        // Асинхронна функція для отримання даних на основі пошукового запиту
        const fetchData = async () => {
            try {
                // Виконуємо запит до API для отримання постів, які містять
пошуковий термін у заголовку
                const response = await
axios.get(`https://04cb5470549a62ec.mokky.dev/posts?title=*${searchTerm}`);

                // Оновлюємо стан даних з отриманими результатами
                setData(response.data);
            } catch (error) {
                // Логування помилки у випадку невдалого запиту
                console.error('Error fetching data:', error);
            }
        };

        // Викликаємо функцію для отримання даних
        fetchData();
    }
}, [searchTerm, initialData]); // Залежності: виконувати ефект при зміні searchTerm
або initialData
```

Рис. 2.2 Фільтрація даних та оновлення стану на основі пошукового запиту

Цей підхід забезпечує зручний та гнучкий механізм пошуку, який значно покращує користувацький досвід при роботі з веб-додатком.

Реалізація фільтрації постів за тегами у веб-додатку

В рамках даної роботи розроблено механізм фільтрації постів за тегами, який дозволяє користувачам ефективно знаходити необхідну інформацію. Цей механізм реалізований за допомогою функцій handleTagToggle, useEffect та handleTagClick.

Функція handleTagToggle (див. Додаток А4) відповідає за додавання або видалення тегів з списку вибраних. Вона приймає tagId як аргумент та оновлює стан tags, змінюючи властивість selected відповідного тега на протилежне.

Функція useEffect виконує дві важливі задачі:

1. **Фільтрація тегів**: При зміні списку тегів, функція useEffect фільтрує список, залишаючи лише ті теги, які були вибрані користувачем. Відфільтрований список зберігається у стані selectedTags.
2. **Збереження стану вибраних тегів**: Функція useEffect також зберігає дані про вибрані теги у локальному сховищі браузера за допомогою localStorage. Це гарантує, що після перезавантаження сторінки, список вибраних тегів буде збережений.

Функція handleTagClick виконує пошук постів за вибраним тегом. Вона приймає тег як аргумент та фільтрує початковий набір даних постів (initialData), залишаючи лише ті, які мають тег, що співпадає з вибраним тегом користувача. Відфільтрований список постів зберігається у стані data, який відповідає за список постів, що відображаються на сторінці.

Завдяки даній реалізації, користувачі можуть легко керувати тегами, додавати та видаляти їх з списку вибраних, а також швидко знаходити необхідні пости за допомогою пошуку за тегами. Цей механізм значно покращує користувацький досвід та робить роботу з веб-додатком більш зручною та ефективною.

```
// Функція для перемикання стану вибору тегу
const handleTagToggle = (tagId) => {
    // Оновлюємо стан тегів, перемикаючи значення 'selected' для відповідного тегу
    setTags((prevTags) =>
        prevTags.map((tag) =>
            tag.id === tagId ? { ...tag, selected: !tag.selected } : tag
        )
    );
};

useEffect(() => {
    // Оновлюємо стан обраних тегів на основі змінених тегів
    setSelectedTags(tags.filter((tag) => tag.selected));

    // Зберігаємо стан вибору тегів у localStorage
    tags.forEach(tag => localStorage.setItem(`tag_${tag.id}`, tag.selected));
}, [tags]); // Виконується кожного разу, коли змінюється стан тегів

// Функція для обробки кліку на тег
const handleTagClick = (tag) => {
    // Фільтруємо початкові дані постів, щоб знайти ті, що містять обраний тег
    const filteredData = initialData.filter(post => post.tags.some(postTag =>
postTag.name === tag.name));

    // Оновлюємо стан даних відфільтрованими постами
    setData(filteredData);
};
```

Рис. 2.3 Управління тегами та фільтрація даних

Цей підхід забезпечує гнучкий та зручний механізм фільтрації, який дозволяє користувачам легко керувати тегами та знаходити потрібну інформацію, значно покращуючи їх взаємодію з веб-додатком.

Реєстрація користувача та обробка даних на сервері

У даній роботі розроблено механізм реєстрації користувачів (див. Додаток Б1), який передбачає відправку даних на сервер для їх обробки. Цей

механізм реалізований за допомогою функції handleSubmit, яка активується при поданні реєстраційної форми.

Функція handleSubmit виконує ряд кроків:

- **Перехоплення події "submit"**: Функція перехоплює подію "submit" форми, щоб запобігти стандартній поведінці браузера, яка призводить до перезавантаження сторінки.
- **Відправлення POST-запиту**: За допомогою бібліотеки Axios, функція handleSubmit формує POST-запит до сервера, передаючи в ньому дані про нового користувача: повне ім'я, електронну пошту та пароль.
- **Обробка відповіді сервера**: У разі успішного виконання запиту, сервер повертає відповідь, яка містить інформацію про результат реєстрації. Ця відповідь виводиться у консоль розробника, а також використовується для автоматичного входу користувача за допомогою функції loginUser.
- **Обробка помилок**: У разі невдалого виконання запиту, функція handleSubmit перехоплює помилку та аналізує її код. Якщо сервер повертає статус 401 (несанкціонований доступ), це означає, що користувач з даною електронною поштою вже зареєстрований. В такому випадку, відображається повідомлення про необхідність авторизації. В інших випадках, відображається загальне повідомлення про помилку реєстрації, а детальна інформація про помилку виводиться у консоль розробника.

```
const handleSubmit = async (e) => {
    // Зупиняємо стандартну поведінку форми (перезавантаження сторінки)
    e.preventDefault();

    try {
        // Виконуємо POST-запит до API для реєстрації нового користувача
        const response = await
axios.post("https://04cb5470549a62ec.mokky.dev/register", {
            fullName: fullName,
```

```
            email: email,
            password: password
        });

        // Якщо реєстрація пройшла успішно, виводимо повідомлення і виконуємо вхід
користувача
        console.log("Registration successful!", response.data);
        loginUser(email, password);
    } catch (error) {
        // Обробляємо помилку
        if (error.response && error.response.status === 401) {
       // Якщо користувач вже зареєстрований, встановлюємо повідомлення про помилку
            setErrorMessage("User already registered. Please log in.");
        } else {
            // Інакше, встановлюємо загальне повідомлення про помилку
            setErrorMessage("Registration failed. Please try again later.");
        }

        // Логуємо додаткові дані про помилку для відладки
        console.error("Registration failed:", error);
    }
};
```

Рис. 2.4 Функція обробки форми реєстрації

Рис. 2.5 Схема обробки форми реєстрації

Завдяки даній реалізації, процес реєстрації користувачів стає автоматизованим та ефективним, гарантуючи безпечну та коректну передачу даних на сервер, а також чітку обробку результатів реєстрації.

Авторизація користувача та обробка даних на сервері

В рамках даної роботи розроблено механізм авторизації користувачів (див. Додаток Б2), який реалізований за допомогою функції handleSubmit. Ця функція активується при поданні форми авторизації та виконує наступні дії:

- **Перехоплення події "submit"**: Функція handleSubmit перехоплює подію "submit" форми, щоб запобігти стандартній поведінці браузера, яка призводить до перезавантаження сторінки.
- **Відправлення POST-запиту**: За допомогою бібліотеки Axios, функція handleSubmit формує POST-запит до сервера, передаючи в ньому дані для авторизації: електронну пошту та пароль користувача.
- **Обробка відповіді сервера**: У разі успішної авторизації, сервер повертає об'єкт userData, який містить інформацію про користувача. Ця інформація зберігається у локальному сховищі браузера за допомогою методу localStorage.setItem, щоб можна було використовувати її пізніше.
- **Перезавантаження сторінки**: Після збереження даних про користувача, сторінка оновлюється за допомогою методу window.location.reload(), відображаючи зміст сторінки у відповідності до успішної авторизації.
- **Обробка помилок**: У разі невдалої авторизації, функція handleSubmit відображає повідомлення про помилку для користувача, а також виводить детальну інформацію про помилку у консоль розробника.

```
const handleSubmit = async (e) => {
    // Забороняємо стандартну поведінку форми (перезавантаження сторінки)
```

```
    e.preventDefault();
    try {
        // Виконуємо POST-запит до API для автентифікації користувача з введеними
email та паролем
        const response = await
axios.post("https://04cb5470549a62ec.mokky.dev/auth", {
            email: email,
            password: password
        });
        // Зберігаємо дані користувача з відповіді
        const userData = response.data;
        // Зберігаємо дані користувача в localStorage
        localStorage.setItem("user", JSON.stringify(userData.data));
        // Виводимо повідомлення про успішну автентифікацію
        console.log("Authentication successful!", userData);
        // Перезавантажуємо сторінку
        window.location.reload();
    } catch (error) {
        // У випадку помилки встановлюємо повідомлення про помилку
        setErrorMessage("Invalid email or password. Please try again.");
        // Логуємо помилку для відладки
        console.error("Authentication failed:", error);
    }
};
```

Рис. 2.6 Функція обробки форми авторизації

Завдяки даній реалізації, процес авторизації користувачів стає автоматизованим та ефективним, гарантуючи безпечну та коректну передачу даних на сервер, а також чітку обробку результатів авторизації.

Відображення інформації про пост та автора

В рамках даної роботи розроблено механізм відображення детальної інформації про пост та автора (див. Додаток В), який використовує хук useEffect та бібліотеку Axios. Цей механізм активується при завантаженні сторінки з ідентифікатором поста.

Функціональність механізму:

1. **Отримання даних про пост**: За допомогою GET-запиту до сервера, функція useEffect отримує дані про пост, використовуючи його ідентифікатор, який передається з URL-адреси.

2. **Отримання даних про автора**: Після отримання даних про пост, функція useEffect виконує ще один GET-запит до сервера, щоб отримати інформацію про користувача, який створив цей пост.
3. **Об'єднання даних**: Отримані дані про пост та користувача об'єднуються в один об'єкт postWithUser. Цей об'єкт містить всю інформацію про пост, а також дані про користувача, який його опублікував.
4. **Відображення інформації**: Об'єкт postWithUser зберігається у стані компонента за допомогою функції setPostWithUser. Це дозволяє веб-додатку відображати всю інформацію про пост на сторінці, включаючи заголовок, текст, зображення, теги, дату публікації, а також ім'я та зображення автора.
5. **Обробка помилок**: У випадку виникнення помилок при отриманні даних з сервера, функція useEffect виводить відповідне повідомлення про помилку у консоль розробника.

```
useEffect(() => {
    // Функція для завантаження посту та даних користувача
    const fetchPostWithUser = async () => {
        try {
            // Виконуємо запит для отримання даних посту за його ID
            const postResponse = await
axios.get(`https://04cb5470549a62ec.mokky.dev/posts/${params.id}`);
            const post = postResponse.data;

            // Виконуємо запит для отримання даних користувача за ID користувача,
що створив пост
            const userResponse = await
axios.get(`https://04cb5470549a62ec.mokky.dev/users/${post.user_id}`);
            const user = userResponse.data;

            // Об'єднуємо дані посту і користувача в один об'єкт
            const postWithUser = {
                id: post.id,
                title: post.title,
                text: post.text,
                image: post.image,
                tags: post.tags,
                date: post.date,
                user: {
```

```
                    id: user.id,
                    fullName: user.fullName,
                    image: user.image
                }
            };

            // Оновлюємо стан з новими даними посту та користувача
            setPostWithUser(postWithUser);
        } catch (error) {
            // Виводимо помилку в консоль, якщо запити не вдалося виконати
            console.error('Error fetching post with user:', error);
        }
    };
    // Викликаємо функцію для завантаження даних посту та користувача
    fetchPostWithUser();
}, [params.id]); // Виконується кожного разу, коли змінюється значення params.id
```

Рис. 2.7 Функція для завантаження посту та даних користувача

Цей механізм забезпечує зручний та гнучкий спосіб відображення детальної інформації про пости та їх авторів, що значно покращує користувацький досвід та робить роботу з веб-додатком більш інформативною та цікавою.

Реалізація створення нового поста у блозі

В рамках даної роботи розроблено механізм створення нових постів, який ґрунтується на функції createPost (див. Додаток Г1). Ця функція збирає дані про новий пост, надіслає їх на сервер та обробляє результат створення.

Функціональність механізму:

1. **Збір даних**: Функція createPost збирає всі необхідні дані для нового поста: заголовок, текст, дату публікації, ідентифікатор автора, зображення та вибрані теги.
2. **Фільтрація тегів**: Список тегів фільтрується, щоб залишалися лише ті, які були обрані користувачем.
3. **Створення об'єкта тегів**: З обраних тегів формується об'єкт selectedTags, який містить їх ідентифікатори та назви.

4. **Підготовка зображення**: Якщо користувач завантажив зображення, воно використовується як зображення поста. В іншому випадку, використовується зображення за замовчуванням.
5. **Відправлення POST-запиту**: За допомогою POST-запиту дані нового поста надсилаються на сервер за вказаною адресою.
6. **Обробка відповіді**: У разі успішного створення поста сервер повертає відповідь, і в консолі виводиться повідомлення про успішне створення. Також встановлюється прапорець redirect, який ініціює перенаправлення користувача на іншу сторінку.
7. **Обробка помилок**: У випадку виникнення помилки під час створення поста, відповідне повідомлення про помилку виводиться у консоль.

```
const createPost = () => {
    // Фільтруємо вибрані теги і створюємо новий масив об'єктів з вибраними тегами
    const selectedTags = tags.filter((tag) => tag.selected).map((tag) => ({
        id: tag.id,
        name: tag.name,
    }));
    // Встановлюємо зображення посту; якщо imageUrl не задано, використовуємо
зображення за замовчуванням
    const postImage = imageUrl ? imageUrl : 'https://i.imgur.com/L29x4vq.png';
    // Виконуємо POST-запит для створення нового посту
    axios.post('https://04cb5470549a62ec.mokky.dev/posts', {
        title: editorTitle.getText(), // Отримуємо заголовок посту з редактора
        text: editor.getHTML(), // Отримуємо текст посту у форматі HTML з редактора
        date: new Date().toISOString(), // Встановлюємо поточну дату у форматі ISO
        user_id: user.id, // Використовуємо ID поточного користувача
        image: postImage, // Використовуємо встановлене зображення
        tags: selectedTags, // Додаємо вибрані теги
    }).then(response => {
        // Якщо пост успішно створено, виводимо відповідне повідомлення
        console.log('Post created successfully:', response.data);
        // Встановлюємо редирект для перенаправлення користувача
        setRedirect(true);
    }).catch(error => {
        // У випадку помилки виводимо повідомлення про помилку
        console.error('Error creating post:', error);
    });
```

```
};
```

Рис. 2.8 Створення нового посту

Цей механізм забезпечує зручний та гнучкий спосіб створення нових постів, даючи користувачам можливість легко публікувати свої думки та ідеї на блозі.

Рис. 2.9 Схема створення нового посту

Реалізація механізму оновлення поста у блозі

В рамках даної роботи розроблено механізм оновлення існуючих постів, який ґрунтується на функції updatePost (див. Додаток Г2). Ця функція збирає оновлені дані про пост, надсилає їх на сервер та обробляє результат оновлення.

Функціональність механізму:

1. **Збір даних**: Функція updatePost збирає всі необхідні дані для оновлення поста: заголовок, текст, зображення та вибрані теги.
2. **Фільтрація тегів**: Список тегів фільтрується, щоб залишалися лише ті, які були обрані користувачем.
3. **Створення об'єкта тегів**: З обраних тегів формується об'єкт selectedTags, який містить їх ідентифікатори та назви.
4. **Підготовка зображення**: Якщо користувач завантажив нове зображення, воно використовується як зображення поста. В іншому випадку, використовується зображення за замовчуванням.
5. **Відправлення PATCH-запиту**: За допомогою PATCH-запиту дані оновленого поста надсилаються на сервер за вказаною адресою.
6. **Обробка відповіді**: У разі успішного оновлення поста сервер повертає відповідь, і в консолі виводиться повідомлення про успішне оновлення. Також встановлюється прапорець redirect, який ініціює перенаправлення користувача на іншу сторінку.
7. **Обробка помилок**: У випадку виникнення помилки під час оновлення поста, відповідне повідомлення про помилку виводиться у консоль.

```
const updatePost = () => {
    // Фільтруємо вибрані теги і створюємо новий масив об'єктів з вибраними тегами
    const selectedTags = tags.filter((tag) => tag.selected).map((tag) => ({
        id: tag.id,
        name: tag.name,
    }));
    // Встановлюємо зображення посту; якщо imageUrl не задано, використовуємо
зображення за замовчуванням
    const postImage = imageUrl ? imageUrl : 'https://i.imgur.com/L29x4vq.png';
```

```
    // Виконуємо PATCH-запит для оновлення посту
    axios.patch(`https://04cb5470549a62ec.mokky.dev/posts/${params.id}`, {
        title: editorTitle.getText(), // Отримуємо заголовок посту з редактора
        text: editor.getHTML(), // Отримуємо текст посту у форматі HTML з редактора
        image: postImage, // Використовуємо встановлене зображення
        tags: selectedTags, // Додаємо вибрані теги
    }).then(response => {
        // Якщо пост успішно оновлено, виводимо відповідне повідомлення
        console.log('Post updated successfully:', response.data);
        // Встановлюємо редирект для перенаправлення користувача
        setRedirect(true);
    }).catch(error => {
        // У випадку помилки виводимо повідомлення про помилку
        console.error('Error updating post:', error);
    });
};
```

Рис. 2.10 Оновлення існуючого посту

Цей механізм забезпечує зручний та гнучкий спосіб оновлення постів, дозволяючи користувачам легко редагувати та оновлювати свої думки та ідеї на блозі.

Додавання нових тегів під час створення поста

В рамках даної роботи розроблено механізм додавання нових тегів до постів, який ґрунтується на функції handleAddTag (див. Додаток Г3). Ця функція активується при введенні нового тегу та надсилає його на сервер для збереження.

Функціональність механізму:

1. **Перевірка введеного тексту**: Функція handleAddTag перевіряє, чи введений користувачем рядок не є порожнім та не складається з пробілів. Якщо введений рядок відповідає цим умовам, виконується наступний крок.
2. **Формування тегу**: Введений тег форматується таким чином, щоб перший символ був великої літери. Це робиться за допомогою методу charAt(0).toUpperCase() + searchTerm.slice(1).

3. **Відправлення POST-запиту**: За допомогою POST-запиту новий тег надсилається на сервер за вказаною адресою. У тілі запиту міститься параметр name з значенням введеного тегу.
4. **Обробка відповіді**: У разі успішного створення тегу сервер повертає статус 201.
5. **Додавання тегу до списку**: Якщо сервер повернув статус 201, новий тег додається до списку тегів у стані компонента за допомогою функції setTags. Ця функція оновлює список тегів, додаючи до нього новий об'єкт з ідентифікатором, назвою, стилем відображення (display: 'flex') та прапорцем selected, встановленим на false.
6. Очищення поля пошуку: Після успішного додавання тегу поле пошуку очищується за допомогою функції setSearchTerm('').
7. **Обробка помилок**: У випадку невдалого створення тегу сервер повертає відповідну помилку, яка виводиться у консоль розробника за допомогою методу console.error.

```
const handleAddTag = async () => {
    // Перевіряємо, чи не є пошуковий запит пустим або складається лише з пробілів
    if (searchTerm.trim() !== '') {
        // Перша літера тега має бути великою, а решта - маленькими
        const capitalizedTagName = searchTerm.charAt(0).toUpperCase() +
searchTerm.slice(1);

        try {
            // Виконуємо POST-запит для додавання нового тега на сервер
            const response = await
axios.post('https://04cb5470549a62ec.mokky.dev/tags', {
                name: capitalizedTagName
            });

            // Перевіряємо, чи тег успішно створений (статус 201)
            if (response.status === 201) {
                // Оновлюємо список тегів додаванням нового тега
                setTags(prevTags => [
                    ...prevTags,
                    {
                        id: response.data.id,
                        name: response.data.name,
```

```
                        display: 'flex', // Додаємо властивість display
                        selected: false // Спочатку тег не вибраний
                    }
                ]);
                // Очищаємо пошуковий запит після додавання тега
                setSearchTerm('');
            } else {
                // Виводимо помилку в консоль, якщо додати тег не вдалося
                console.error('Failed to add tag:', response);
            }
        } catch (error) {
            // Логування помилки у випадку виникнення помилки під час запиту
            console.error('Error adding tag:', error);
        }
    }
};
```

Рис. 2.11 Додавання нового тегу

Цей механізм значно покращує функціональність створення постів, даючи користувачам можливість легко додавати нові теги до своїх публікацій, роблячи їх більш організованими та зручними для пошуку.

Пошук зображень на Unsplash

Unsplash - це безкоштовний онлайн-ресурс, який пропонує широкий спектр високоякісних зображень, які можна використовувати для різних цілей, таких як веб-дизайн, блоги, презентації та багато іншого. Його колекція постійно поповнюється завдяки роботам професійних фотографів та художників з усього світу.

Переваги використання Unsplash:

- **Безкоштовні зображення**: Unsplash надає безкоштовний доступ до мільйонів зображень високої роздільної здатності, які можна використовувати без будь-яких обмежень.
- **Висока якість**: Всі зображення на Unsplash проходять ретельний відбір, щоб гарантувати їх високу якість та естетичну цінність.

- **Різноманіття**: Колекція Unsplash охоплює широкий спектр тем та категорій, що дозволяє легко знайти зображення, які відповідають вашим потребам.
- **Простота використання**: Unsplash має зручний інтерфейс, який полегшує пошук та завантаження зображень.
- **Ліцензія Creative Commons Zero** (CC0): Зображення Unsplash ліцензуються за Creative Commons Zero (CC0), що дає користувачам право використовувати їх без будь-яких обмежень, включаючи модифікацію та комерційне використання.

В рамках даної роботи розроблено механізм пошуку зображень на популярному сервісі Unsplash (див. Додаток Г4), який ґрунтується на функції searchUnsplashImages. Ця функція використовує API Unsplash для пошуку зображень за введеним користувачем запитом.

Функціональність механізму:

1. **Відправлення GET-запиту**: Функція searchUnsplashImages використовує метод GET для надсилання запиту до API Unsplash. У запиті містяться два параметри:
   - query: Запит користувача, за яким буде відбуватися пошук зображень.
   - client_id: Ключ API користувача, отриманий при реєстрації на Unsplash.
2. **Обробка відповіді**: У разі успішного виконання запиту, API Unsplash повертає JSON-відповідь, яка містить масив результатів пошуку. Цей масив містить інформацію про знайдені зображення, включаючи їх URL-адреси, описи, розміри та ін.
3. **Зберігання результатів**: Отримані результати пошуку зберігаються у стані компонента за допомогою функції

setSearchResults. Це робить їх доступними для відображення на сторінці.

4. **Обробка помилок**: У випадку виникнення помилок під час пошуку зображень, відповідне повідомлення про помилку виводиться у консоль розробника за допомогою методу console.error.

```
const searchUnsplashImages = async () => {
    try {
        // Виконуємо GET-запит до API Unsplash для пошуку зображень за пошуковим
запитом
        const response = await
axios.get(`https://api.unsplash.com/search/photos?query=${searchQuery}&client_id=d7
5A_CGFOPD6kE5oc7dX-UEODZ6hzAUb-O6Z0USyGXw`);
        // Оновлюємо стан результатами пошуку
        setSearchResults(response.data.results);
    } catch (error) {
        // Логування помилки у випадку виникнення помилки під час запиту
        console.error('Error searching images:', error);
    }
};
```

Рис. 2.12 Пошук зображень на Unsplash

Цей механізм значно розширює функціональність веб-додатку, даючи користувачам доступ до величезної колекції безкоштовних зображень високої якості з Unsplash, що робить його цінним інструментом для дизайнерів, блогерів, маркетологів та інших користувачів.

3.3 Додаткові бібліотеки та інструменти

TipTap: Розширення можливостей веб-редакторів тексту з Vue.js

В рамках даної роботи досліджено можливості бібліотеки TipTap, яка є інноваційним інструментом для роботи з текстовим вмістом у веб-додатках на основі Vue.js. TipTap відкриває нові горизонти для створення багатофункціональних та інтерактивних редакторів тексту, що робить його цінним інструментом для розробників.

Функціональні можливості TipTap:

- **Формування тексту**: TipTap надає широкий спектр інструментів для форматування тексту, включаючи зміну шрифту, розміру, кольору, жирного, курсиву, підкреслення та ін.
- **Вставлення зображень**: Користувачі можуть легко вставляти зображення в текст, налаштовувати їх розмір, положення та інші параметри.
- **Списки та таблиці**: TipTap дозволяє створювати структуровані списки та таблиці, що робить текст більш організованим та зручним для сприйняття.
- **Модульна архітектура**: TipTap має модульну архітектуру, що дозволяє розробникам легко додавати нові функції та розширювати можливості редактора відповідно до своїх потреб.
- **Підтримка JavaScript та TypeScript**: TipTap підтримує JavaScript та TypeScript, що робить його доступним для широкого кола розробників.
- **Активна спільнота користувачів**: TipTap має активну спільноту користувачів, які надають підтримку, діляться знаннями та досвідом, а також сприяють розвитку бібліотеки.

Переваги використання TipTap:

- **Створення інтерактивних редакторів**: TipTap дає можливість створювати інтерактивні редактори тексту, які дозволяють користувачам легко форматувати текст, вставляти зображення, створювати списки та таблиці, а також використовувати інші функції.
- **Зручний та привабливий інтерфейс**: TipTap дозволяє розробникам створювати зручні та привабливі інтерфейси для редакторів тексту, що робить роботу з текстом більш комфортною та приємною.

- **Розширюваність**: Модульна архітектура TipTap робить його легко розширюваним, що дозволяє розробникам додавати нові функції та адаптувати редактор до своїх конкретних потреб.
- **Доступність**: Підтримка JavaScript та TypeScript робить TipTap доступним для широкого кола розробників.
- **Підтримка спільноти**: Активна спільнота користувачів TipTap забезпечує підтримку, знання та досвід, що робить роботу з бібліотекою більш ефективною.
- **Регулярні оновлення**: TipTap постійно оновлюється, що гарантує його актуальність та відповідність новим технологіям.

Бібліотека TipTap є потужним та гнучким інструментом для створення інтерактивних та багатофункціональних редакторів тексту на основі Vue.js. Його широкі можливості, модульна архітектура, активна спільнота користувачів та регулярні оновлення роблять TipTap цінним вибором для розробників, які прагнуть створювати високоякісні веб-додатки з розширеними можливостями роботи з текстовим вмістом.

Створення інтерфейсу для написання постів з TipTap

В рамках даної роботи досліджено можливості бібліотеки TipTap для створення інтерфейсу написання постів у веб-додатку. TipTap дає можливість розробникам створювати потужні та гнучкі текстові редактори, що робить його цінним інструментом для даної задачі (див. Додаток Г1).

Реалізація інтерфейсу:

1. **Редактор заголовка**:
    - Використовується useEditor з розширеннями StarterKit та Placeholder для налаштування редактора заголовка.
    - Параметр content визначає початковий вміст редактора, у нашому випадку - порожній заголовок.

```
// Ініціалізація редактора для заголовку поста
const editorTitle = useEditor({
    // Налаштування розширень для редактора
    extensions: [
        // Базовий набір розширень, включаючи основні елементи для редагування
        StarterKit,
        // Розширення для відображення підказки (placeholder)
        Placeholder.configure({
            placeholder: 'Заголовок', // Текст підказки
        })
    ],
    // Початковий контент редактора
    content: `
    <h1></h1>
`,
});
```

Рис. 3.1 Ініціалізація редактора для заголовку поста

2. **Основний редактор**:
    - Використовується useEditor з розширеннями StarterKit, Document, Paragraph, Text, Image та Dropcursor для створення основного редактора.
    - Цей редактор надає користувачам можливість вставляти текст, зображення та інші елементи у свої пости.
    - Placeholder використовується для вказівки користувачам початкового тексту.

```
// Ініціалізація основного редактора для контенту поста
const editor = useEditor({
    // Налаштування розширень для редактора
    extensions: [
        // Базовий набір розширень, включаючи основні елементи для редагування
        StarterKit,
        // Розширення для документа (структурний елемент)
        Document,
        // Розширення для параграфів (структурний елемент)
        Paragraph,
        // Розширення для тексту
        Text,
        // Розширення для зображень
        Image,
        // Розширення для показу курсору при перетягуванні елементів
        Dropcursor,
        // Розширення для відображення підказки (placeholder)
```

```
        Placeholder.configure({
            placeholder: 'Почніть писати тут...', // Текст підказки
        })
    ],
    // Початковий контент редактора
    content: `
    <p></p>
`,
});
```

Рис. 3.2 Ініціалізація основного редактора для контенту поста

Використання бібліотеки TipTap для створення інтерфейсу написання постів дає ряд переваг, таких як потужність, гнучкість, зручність та розширюваність. TipTap є цінним інструментом для розробників, які прагнуть створювати веб-додатки з розширеними можливостями для написання та редагування контенту.

Редагування стилів тексту в інтерфейсі редактора

В рамках даної роботи досліджено можливості реалізації функціоналу редагування стилів тексту в інтерфейсі редактора (див. Додаток Г4). Цей функціонал дозволяє користувачам змінювати форматування тексту, роблячи процес написання та редагування постів більш зручним та ефективним.

Реалізація:

1. BubbleMenu:
    - Використовується компонент BubbleMenu, який відображається поруч з виділеним текстом.
    - BubbleMenu активується тільки тоді, коли користувач виділяє текст.
2. Кнопки форматування:
    - У BubbleMenu розміщені кнопки для зміни стилів тексту: жирний, курсив та заголовок першого рівня.
    - При натисканні на кнопку, відповідна команда виконується у редакторі.

```
{editor && (
    // Відображаємо меню BubbleMenu, якщо редактор ініціалізовано
    <BubbleMenu
        className="bubble-menu"
        tippyOptions={{ duration: 100 }} // Налаштування для Tippy.js
        editor={editor} // Прив'язуємо редактор до меню
    >
        {/* Кнопка для переключення жирного тексту */}
        <button
            onClick={() => editor.chain().focus().toggleBold().run()} // Виконує
команду для переключення жирного тексту
            className={editor.isActive('bold') ? 'is-active' : ''} // Встановлює
клас 'is-active', якщо жирний текст активний
        >
            <BoldIcon /> {/* Іконка для жирного тексту */}
        </button>
        {/* Кнопка для переключення курсивного тексту */}
        <button
            onClick={() => editor.chain().focus().toggleItalic().run()} // Виконує
команду для переключення курсивного тексту
            className={editor.isActive('italic') ? 'is-active' : ''} // Встановлює
клас 'is-active', якщо курсивний текст активний
        >
            <ItalicIcon /> {/* Іконка для курсивного тексту */}
        </button>
        {/* Кнопка для переключення заголовку рівня 1 */}
        <button
            onClick={() => editor.chain().focus().toggleHeading({ level: 1
}).run()} // Виконує команду для переключення заголовку рівня 1
            className={editor.isActive('heading', { level: 1 }) ? 'is-active' : ''}
// Встановлює клас 'is-active', якщо заголовок рівня 1 активний
        >
            <H1Icon /> {/* Іконка для заголовку рівня 1 */}
        </button>
    </BubbleMenu>
)}
```

Рис. 3.3 Відображаємо меню BubbleMenu

Переваги:

- **Зручність використання**: Користувачі можуть легко змінювати стилі тексту за допомогою кнопок у BubbleMenu.
- **Ефективність**: Редагування стилів тексту економить час та зусилля користувачів.
- **Гнучкість**: Можна додавати кнопки для інших стилів тексту, таких як підкреслення, вирівнювання, шрифт та розмір.

- **Доступність**: BubbleMenu активується тільки тоді, коли користувач виділяє текст, що робить його зручним для використання.

Реалізація функціоналу редагування стилів тексту в інтерфейсі редактора за допомогою BubbleMenu та кнопок форматування значно покращує зручність та ефективність роботи користувачів з текстом. Це робить інтерфейс редактора більш гнучким та доступним, що робить його цінним інструментом для створення та редагування контенту.

Реалізація роутінгу в веб-додатку React

В рамках даної роботи досліджено можливості бібліотеки react-router-dom для реалізації роутінгу в веб-додатку React. Роутінг є важливою частиною веб-додатків, адже він забезпечує навігацію користувача між сторінками та покращує їх користувальницький досвід.

Функція App є основним компонентом React-додатку, де й відбувається реалізація роутінгу. Вона використовує такі ключові моменти:

1. **Визначення поточного маршруту:**
    - За допомогою хука useLocation отримується інформація про поточне розташування користувача в додатку.
    - З змінної location витягується pathname, який представляє шлях до поточної сторінки.
    - Використовується умовний оператор, який порівнює pathname з можливими значеннями шляхів, що відповідають першій сторінці додатку.
2. **Визначення маршрутів:**
    - Компонент <Routes> використовується для декларування маршрутів у додатку.
    - Кожен маршрут визначається за допомогою компоненту <Route>, де:
    - path: шлях до маршруту.

- element: компонент, який буде відображатися на цій сторінці.

3. **Умовне відображення футера**
   - Футер відображається на всіх сторінках, окрім першої.

Опис маршрутів:

- **/falts/ або /falts**: Цей маршрут відповідає за відображення компоненту FirstVisit, який зазвичай використовується для першого знайомства користувача з веб-додатком.
- **/falts/home**: Цей маршрут відповідає за головну сторінку додатку, де відображається компонент Home.
- **/falts/userPosts**: Цей маршрут відповідає за сторінку користувацьких постів, де відображається компонент UserPosts.
- **/falts/post/:id**: Цей маршрут відповідає за сторінку конкретного поста, де :id - це ID поста. На цій сторінці відображається компонент FullPost.
- **/falts/write**: Цей маршрут відповідає за сторінку створення нового поста, де відображається компонент CreatePost.
- **/falts/edit/post/:id**: Цей маршрут відповідає за сторінку редагування поста, де :id - це ID поста. На цій сторінці відображається компонент EditPost.

Переваги використання роутінгу:

- **Зручна навігація**: Користувачі можуть легко переміщатися між сторінками веб-додатку за допомогою URL-адрес.
- **Організована структура**: Роутінг чітко структурує веб-додаток, роблячи його більш зрозумілим та зручним для користувачів.
- **Модульність**: Роутінг дозволяє розділити веб-додаток на логічні частини, що робить його код більш організованим та легким для обслуговування.

- **Покращений UX**: Завдяки зручній навігації та чіткій структурі роутінг робить веб-додаток більш приємним у використанні для користувачів.

Реалізація роутінгу в React-додатку за допомогою бібліотеки react-router-dom є важливою практикою, яка робить його більш зручним, організованим та приємним у використанні. Роутінг покращує навігацію, структуру та загальний користувальницький досвід (UX) веб-додатку.

```
function App() {
  const location = useLocation();
  const { pathname } = location;
  // Перевіряємо, чи користувач першій сторінці
  const firstPage = pathname === '/falts/' || pathname === '/falts';
  return (
    <>
      {/* Використовуємо компоненти маршрутизації для відображення відповідних
компонентів для кожного шляху */}
      <Routes>
        {/* Компонент для першого відвідування */}
        <Route path='/falts/' element={<FirstVisit />} />
        {/* Головна сторінка */}
        <Route path='/falts/home' element={<Home />} />
        {/* Сторінка з постами користувача */}
        <Route path='/falts/userPosts' element={<UserPosts />} />
        {/* Повний пост */}
        <Route path='/falts/post/:id' element={<FullPost />} />
        {/* Сторінка для створення нового поста */}
        <Route path='/falts/write' element={<CreatePost />} />
        {/* Сторінка для редагування існуючого поста */}
        <Route path='/falts/edit/post/:id' element={<EditPost />} />
      </Routes>
      {/* Відображаємо підвал, якщо користувач не першій сторінці */}
      {!firstPage && <footer />}
    </>
  );
}
```

Рис. 3.4 Реалізація роутінгу в React-додатку

У наведеному коді використовується хук useLocation з бібліотеки react-router-dom. Цей хук надає доступ до інформації про поточне розташування користувача в додатку, що використовується для визначення активного маршруту та відображення відповідного компоненту.

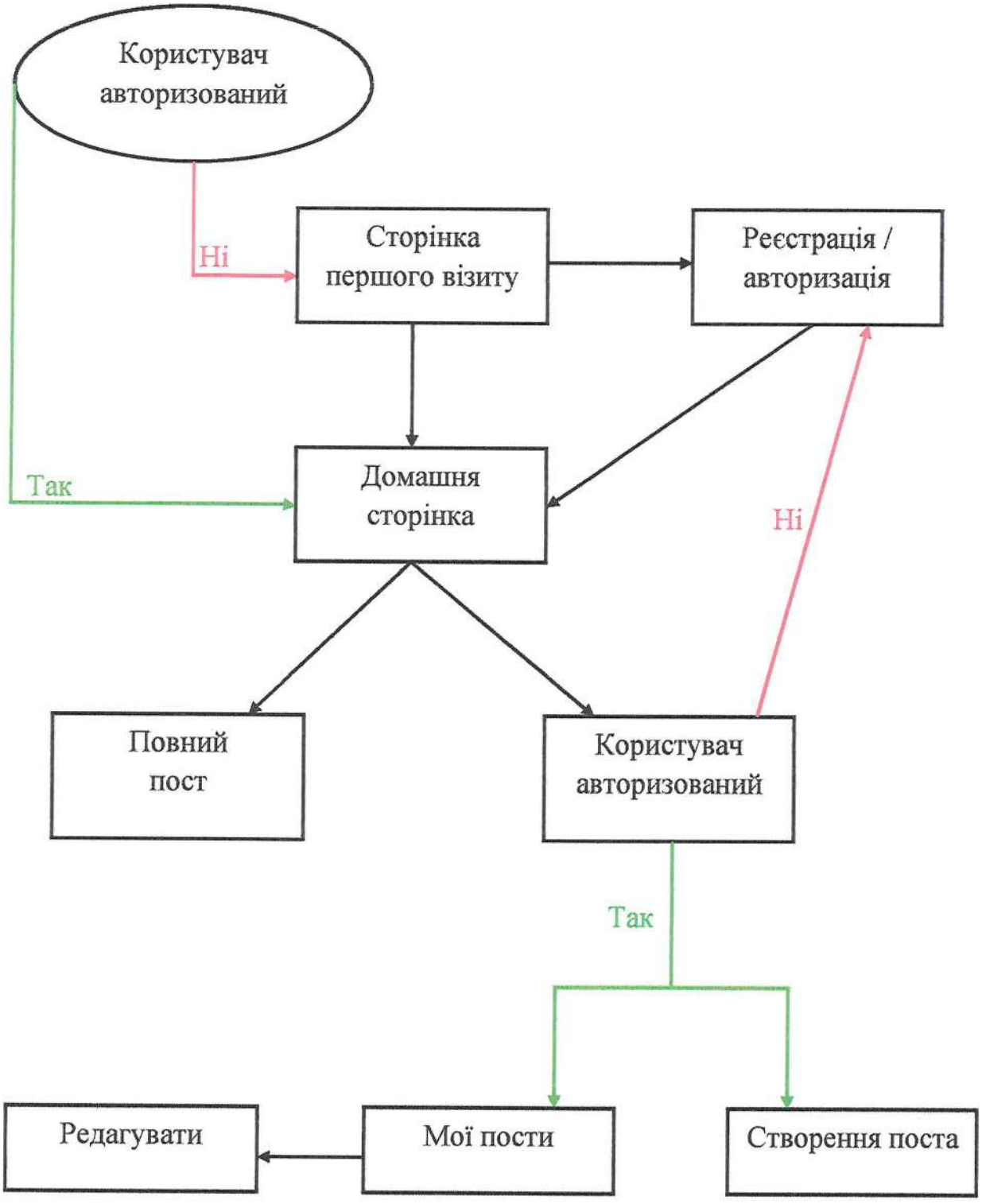


Рис. 3.5 Схема роутінгу в React-додатку

ВИСНОВОК

У ході дипломної роботи було розроблено сайт блог платформу з використанням React. Платформа дозволяє користувачам створювати та публікувати статті, коментарі, підписуватися на інших користувачів та отримувати сповіщення про нові записи.

В процесі розробки сайту було використано такі технології:

- React: JavaScript бібліотека для декларативного створення інтерфейсів користувача.
- MockAPI.io: сервіс для емуляції API.

Було розроблено наступні функціональні можливості:

- Створення та публікація статей.
- Пошук статей.
- Перегляд популярних статей.
- Перегляд статей за категоріями.
- Редагування та видалення статей.

Було проведено тестування сайту з метою виявлення та виправлення помилок. Сайт доступний за адресою https://ururu2407.github.io/falts/ .

Розроблена сайт блог платформа є зручним та функціональним інструментом для створення та ведення блогів. Платформа може бути використана як приватними користувачами, так і організаціями.

Розроблений сайт блог платформа може бути використаний для:

- Створення особистих блогів.
- Створення корпоративних блогів.
- Створення новинних сайтів.
- Створення форумів.

- Створення сайтів з відгуками.

Код сайту можна переглянути в Додатку Д.

Дипломна робота є актуальним дослідженням в галузі розробки веб-застосунків. Розроблена сайт блог платформа є зручним та функціональним інструментом для створення та ведення блогів. Платформа має широкі перспективи подальшого розвитку та може бути використана в різних сферах діяльності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Statista. [Електронний ресурс] // Режим доступу до ресурсу: https://www.statista.com/markets/424/topic/539/reach-traffic/
2. State of JavaScript. [Електронний ресурс] // Режим доступу до ресурсу: https://2022.stateofjs.com/ua-UA/
3. MDN Web Docs JavaScript Usage Statistics. [Електронний ресурс] // Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/JavaScript
4. A Brief History of HTML, CSS and JavaScript W3Schools. [Електронний ресурс] // Режим доступу до ресурсу: https://www.w3schools.in/html/history
5. JavaScript's Influence on Web Development. [Електронний ресурс] // Режим доступу до ресурсу:https://plavno.io/company/blog/javascript-full-stack-powerhouse
6. Asynchronous JavaScript and XML (AJAX).[Електронний ресурс] // Режим доступу до ресурсу:https://en.wikipedia.org/wiki/Ajax_(programming)
7. JavaScript Object Notation (JSON). [Електронний ресурс] // Режим доступу до ресурсу:https://www.json.org/
8. Web Services. [Електронний ресурс] // Режим доступу до ресурсу:https://www.w3schools.com/xml/xml_services.asp
9. HyperText Markup Language 5 (HTML5). [Електронний ресурс] // Режим доступу до ресурсу: https://www.w3schools.com/html/html5_intro.asp
10. Cascading Style Sheets 3 (CSS3). [Електронний ресурс] // Режим доступу до ресурсу: https://www.w3schools.com/css/default.asp
11. Angular. [Електронний ресурс] // Режим доступу до ресурсу: https://angular.io/
12. React. [Електронний ресурс] // Режим доступу до ресурсу: https://reactjs.org/
13. Vue.js. [Електронний ресурс] // Режим доступу до ресурсу: https://vuejs.org/

14. Node.js. [Електронний ресурс] // Режим доступу до ресурсу: https://nodejs.org/
15. Python. [Електронний ресурс] // Режим доступу до ресурсу: https://www.python.org/
16. PHP. [Електронний ресурс] // Режим доступу до ресурсу: https://www.php.net/
17. Java. [Електронний ресурс] // Режим доступу до ресурсу: https://www.java.com/en/
18. MySQL. [Електронний ресурс] // Режим доступу до ресурсу: https://www.mysql.com/
19. PostgreSQL. [Електронний ресурс] // Режим доступу до ресурсу:https://www.postgresql.org/
20. MongoDB. [Електронний ресурс] // Режим доступу до ресурсу:https://www.mongodb.com/
21. Git. [Електронний ресурс] // Режим доступу до ресурсу:https://git-scm.com/
22. Scrum. [Електронний ресурс] // Режим доступу до ресурсу: https://www.atlassian.com/agile/scrum
23. Agile Methodologies. [Електронний ресурс] // Режим доступу до ресурсу: https://agilemanifesto.org/
24. HTTPS. [Електронний ресурс] // Режим доступу до ресурсу: https://en.wikipedia.org/wiki/HTTPS
25. SSL/TLS. [Електронний ресурс] // Режим доступу до ресурсу: https://aws.amazon.com/ru/what-is/ssl-certificate/
26. OWASP. [Електронний ресурс] // Режим доступу до ресурсу: https://owasp.org/www-project-top-ten/
27. A Comparative Look at JavaScript Libraries and Frameworks.[Електронний ресурс] // Режим доступу до ресурсу: https://www.sencha.com/blog/js-frameworks/

28. The 40 Best JavaScript Libraries and Frameworks. [Електронний ресурс] // Режим доступу до ресурсу: https://kinsta.com/javascript/

29. The History of React.js on a Timeline. [Електронний ресурс] // Режим доступу до ресурсу: https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/

30. Choosing Between Class and Functional Components in React. [Електронний ресурс] // Режим доступу до ресурсу: https://medium.com/@supraja_miryala/functional-components-vs-class-components-in-react-e2b8aec3ed64

31. Axios Documentation. [Електронний ресурс] // Режим доступу до ресурсу: https://axios-http.com/

ДОДАТКИ

Додаток А

Рис. А1

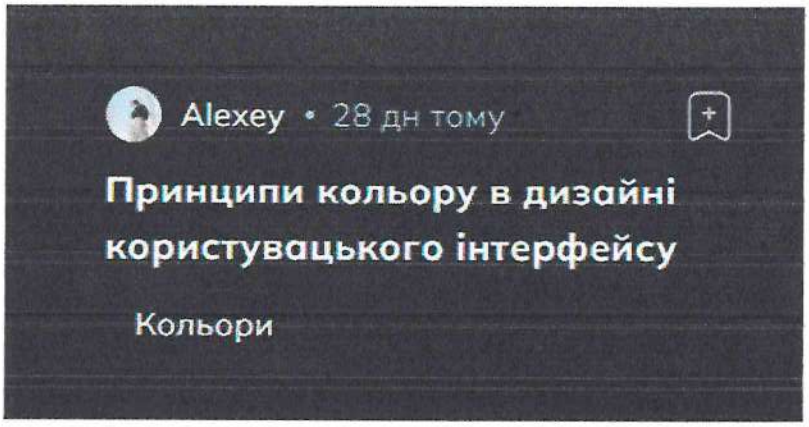


Рис. А2

Рис. А3

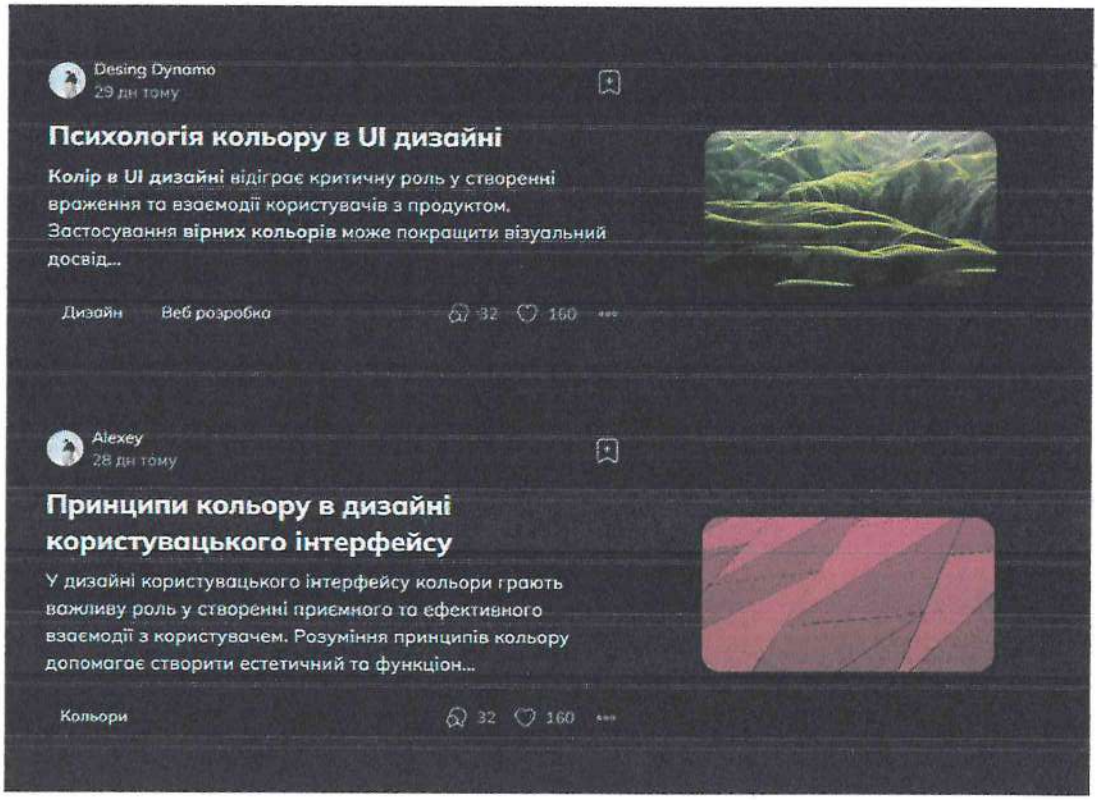
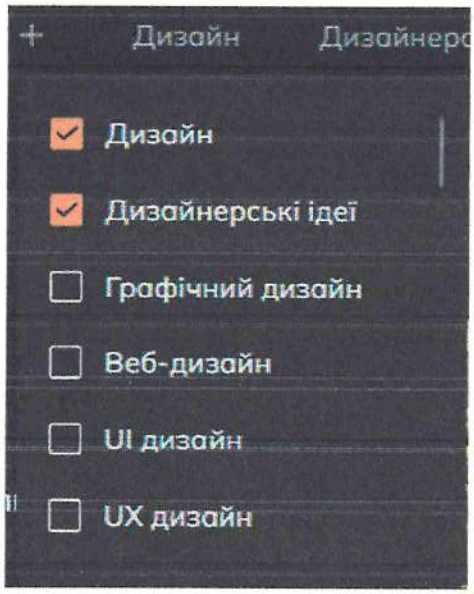


Рис. А4


Дизайн
Дизайнерс
Дизайн
Дизайнерські ідеї
Графічний дизайн
Веб-дизайн
UI дизайн
UX дизайн

Додаток Б

Рис. Б1

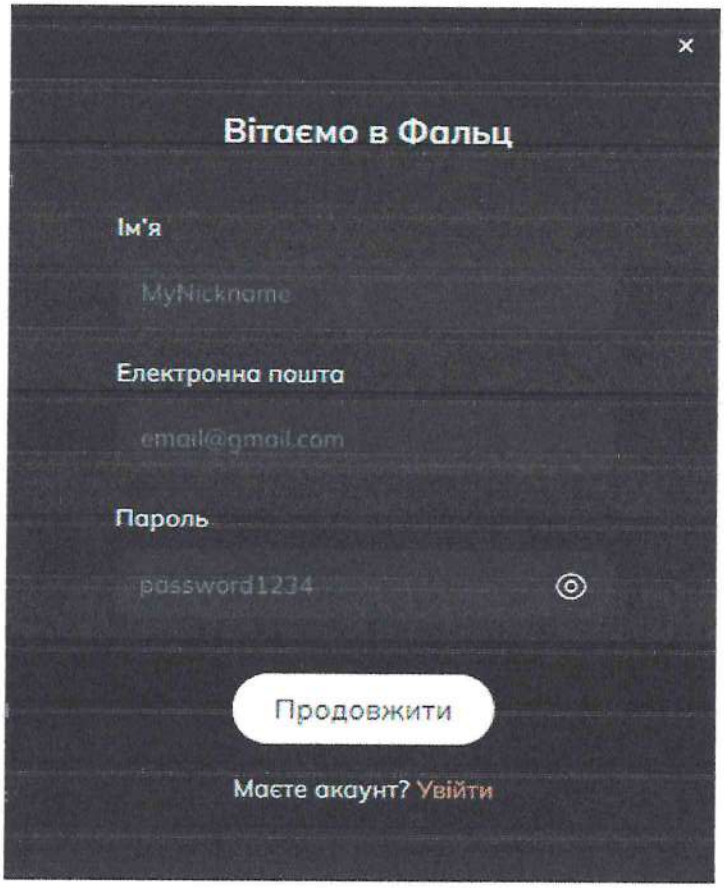
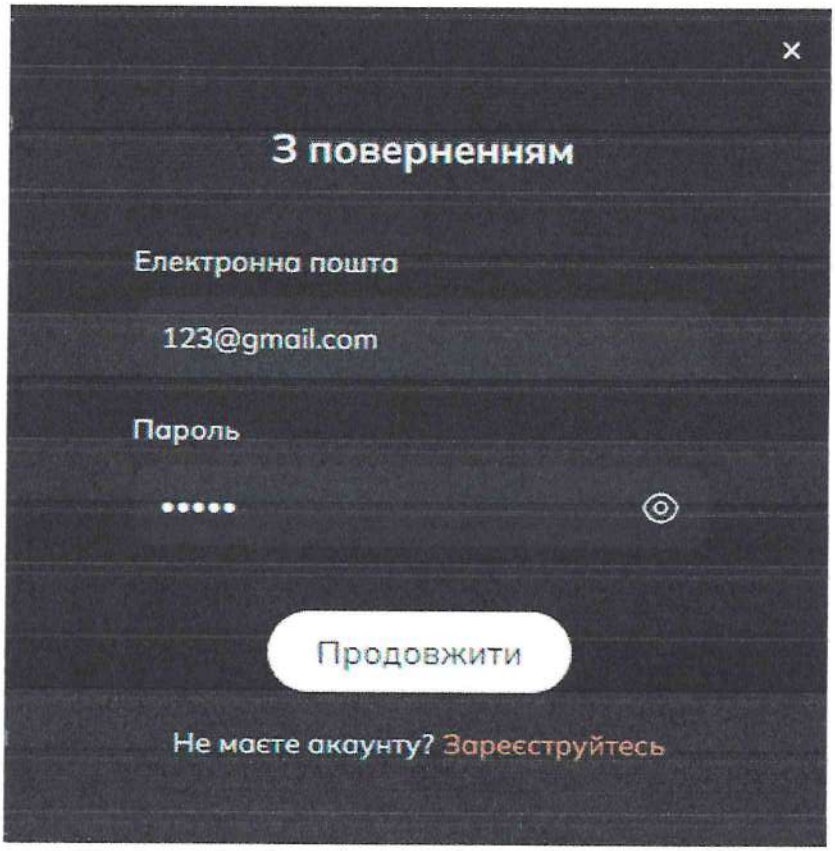
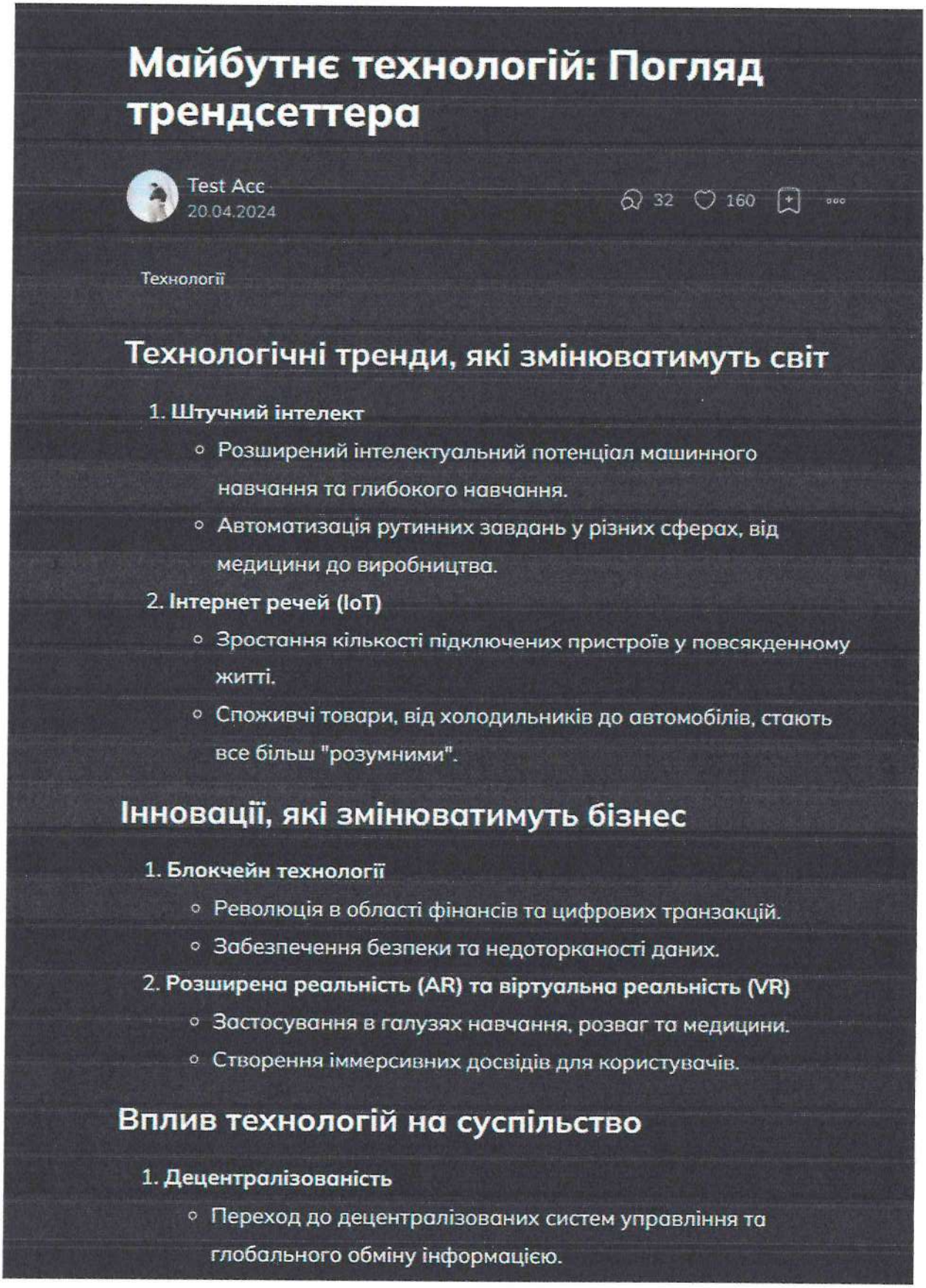


Рис. Б2


З поверненням
Електронна пошта
123@gmail.com
Пароль
Продовжити
Не маєте акаунту? Зареєструйтесь

Додаток В



Майбутнє технологій: Погляд трендсеттера

Test Acc
20.04.2024

32 160

Технології

Технологічні тренди, які змінюватимуть світ

1. **Штучний інтелект**
   - Розширений інтелектуальний потенціал машинного навчання та глибокого навчання.
   - Автоматизація рутинних завдань у різних сферах, від медицини до виробництва.
2. **Інтернет речей (IoT)**
   - Зростання кількості підключених пристроїв у повсякденному житті.
   - Споживчі товари, від холодильників до автомобілів, стають все більш "розумними".

Інновації, які змінюватимуть бізнес

1. **Блокчейн технології**
   - Революція в області фінансів та цифрових транзакцій.
   - Забезпечення безпеки та недоторканості даних.
2. **Розширена реальність (AR) та віртуальна реальність (VR)**
   - Застосування в галузях навчання, розваг та медицини.
   - Створення іммерсивних досвідів для користувачів.

Вплив технологій на суспільство

1. **Децентралізованість**
   - Перехід до децентралізованих систем управління та глобального обміну інформацією.

Додаток Г

Рис. Г1

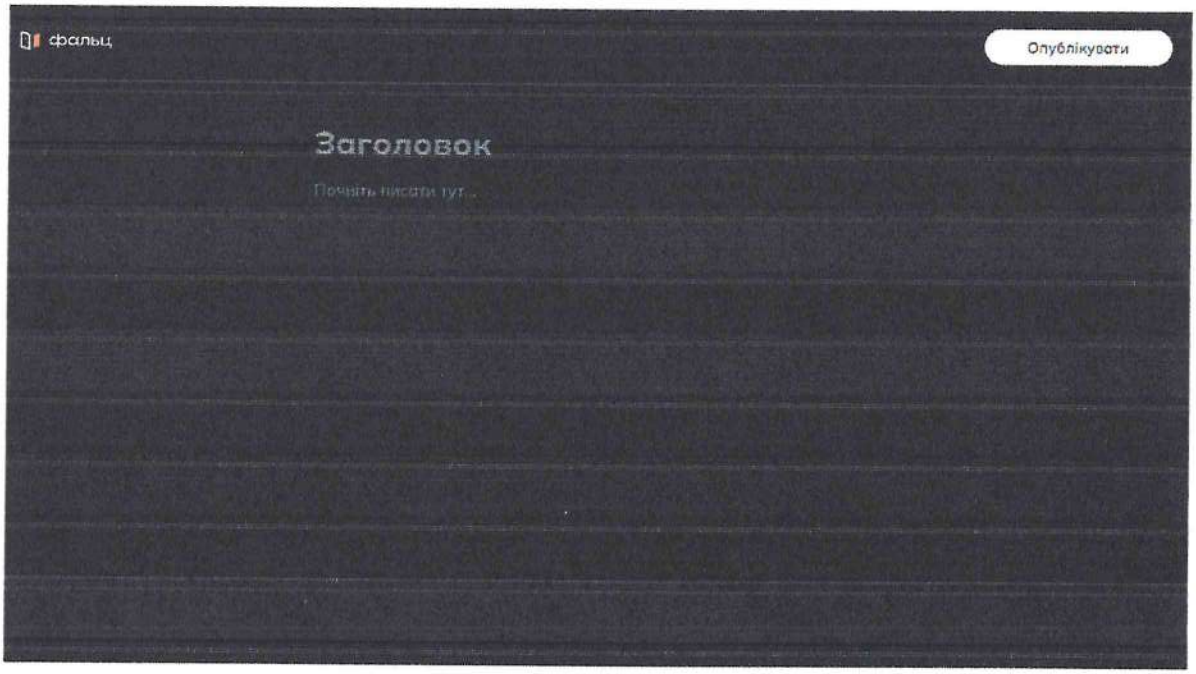


Рис. Г2

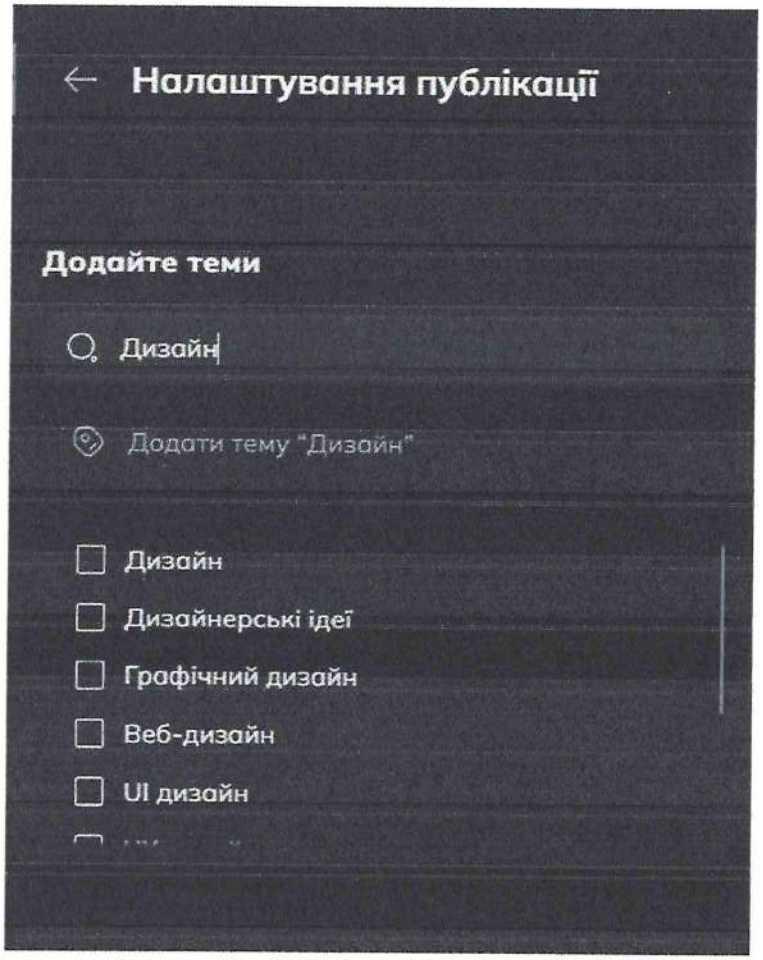
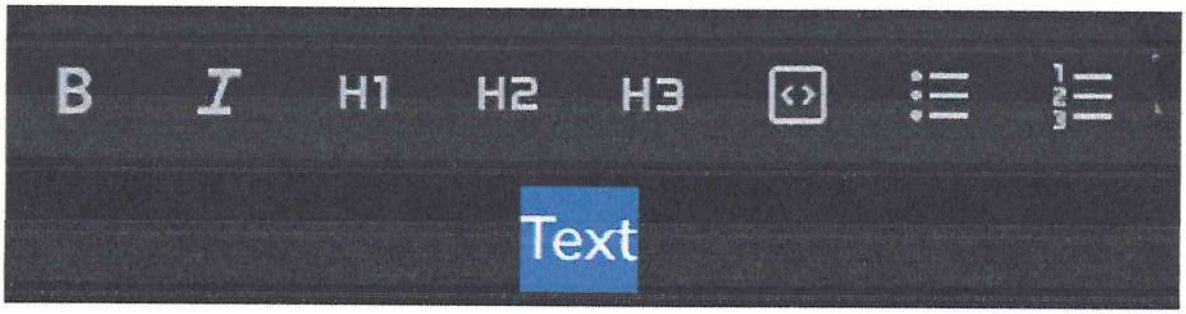


Рис. Г3

Рис. Г4



Додаток Д