МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Факультет математики і інформатики

Кафедра теоретичної та прикладної інформатики

Кваліфікаційна робота

бакалавр

на тему «Проектування та розробка користувацької частини для сайту Студабр»

Виконав: студент 4 курсу, групи МФ-41

спеціальність 122 «Комп’ютерні науки»

освітньо-професійна програма «Теоретична та прикладна інформатика»

Кольєв А. С.

Керівник Бережна Н. І.

Рецензент

Харків – 2024 року

ЗМІСТ

ВСТУП.....3

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ ДО КОРИСТУВАЦЬКОЇ ЧАСТИНИ САЙТУ СТУДАБР.....5

1.1. Формулювання задачі та обґрунтування актуальності теми ……..5

1.2. Аналіз технологій, що використовувались ……..5

1.3. Аналіз функціоналу, який потрібно розробити ……14

2. ПРОЕКТУВАННЯ КОРИСТУВАЦЬКОЇ ЧАСТИНИ САЙТУ СТУДАБР……..……15

2.1. Архітектурні рішення та структура системи ……15

2.2. Опис та конфігурація сховища станів ……17

2.3. Опис конфігурації бібліотеки Axios……..22

2.4. Опис сервісів для взаємодії з API……22

3. РОЗРОБКА КОРИСТУВАЦЬКОЇ ЧАСТИНИ САЙТУ СТУДАБР ……24

3.1. Вибір та налаштування середовища розробки ……...24

3.2. Реалізація основних сторінок та модулів……...28

4. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ КОРИСТУВАЦЬКОЇ ЧАСТИНИ САЙТУ СТУДАБР.……..51

4.1. Методи та інструменти тестування ……51

4.2. Результати тестування ……..52

4.3. Впровадження та розгортання користувацької частини ……...55

ВИСНОВОК ……57

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ……...59

ВСТУП

Кожен день сучасної людини так чи інакше пов’язан з різноманітними технологіями, інтернет-застосунками, соціальними мережами тощо. Завдяки цьому людина отримує безліч інформації щодня, стає розумнішою та більш розвиненою. Але цієї інформації занадто багато, і серед неї дуже багато непотрібного сміття. Тут і приходить надопомогу платформа Студабр, яка ставить собі за мету обмін корисними знаннями між користувачами інтернету. Будь-яка людина може поділитися чимось корисним з іншими, створюючи публікації на нашій платформі, а також навчитися чомусь новому від інших користувачів! Неважливо,хто ти та чим займаєшся, головне, що ти завжди зможеш знайти на Студабр щось цікаве для себе.

Мета і завдання дослідження

Метою цієї роботи є розробка користувацької частини сайту Студабр, яка забезпечить зручне, стабільне й ефективне рішення для публікації та взаємодії зі статтями. Основні завдання дослідження включають:

1. Аналіз предметної області та вимог до користувацької частини сайту Студабр.
2. Проектування користувацької частини сайту Студабр.
3. Створення та налаштування конфігураційних файлів.
4. Розробка користувацької частини сайту Студабр.
5. Розробка текстового редактора для сайту Стубадр.
6. Створення сучасного та зручного інтерфейсу для сайту Студабр.
7. Тестування користувацької частини сайту Студабр.
8. Впровадження та розгортання сайту Студабр.

Актуальність теми

Ця тема є актуальною через велику потребу в безкоштовних освітніх інтернет-ресурсах та самоосвіті. Через величезний обсяг інформації в інтернеті, доступ до корисної та потрібної сильно ускладнюється. В умовах постійного розвитку інформаційного суспільства, доступ до якісної інформації стає просто необхідним. Студабр задовольняє цю потребу, надаючи зручний інструмент для створення та поширення знань.

Методи дослідження

Для досягнення поставленої мети використані такі методи дослідження:

1. Аналіз літературних джерел: Вивчення наукових публікацій та технічної документації, проходження пов’язаних з розробкою сайтів курсів.
2. Проектування системи: Використання компонентно-орієнтованої архітектури для створення користувацької частини.
3. Розробка програмного забезпечення: Використання мови програмування TypeScript, бібліотеки React та фреймворку Bootstrap для реалізаціїкористувацької логіки.
4. Тестування: Застосування методів модульного та інтеграційного тестування для перевірки коректності роботи користувацької частини.

Застосування цих методів дозволило забезпечити достовірність отриманих результатів, а також забезпечити надійне, швидке та якісне функціонування користувацької частини сайту Студабр!

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ ДО КОРИСТУВАЦЬКОЇ ЧАСТИНИ САЙТУ СТУДАБР

1.1. Формулювання задачі та обґрунтування актуальності теми

Задачею даної дипломної роботи є розробка користувацької частини сайту Студабр, що забезпечить зручне, стабільне та ефективне функціонування платформи для створення та взаємодії з публікаціями, доданими іншими користувачами. Сайт має стати ефективним та зручним інструментом для обміну знаннями між користувачами. Це потребує розробки сучасних компонентів та сторінок, які будуть інтуітивно зрозумілими і зовнішньо приємні, а також впровадження ефективної логіки для обробки даних, їх перетворення, валідації та передачі на серверну частину.

Актуальність теми обумовлена зростаючою потребою в якісних освітніх ресурсах, безкоштовних платформах та недостатньою кількостю та якостю існуючих платформ для самоосвіти.

1.2. Аналіз технологій, що використовувались

Розробка користувацької частини сайту Студабр вимагає використання сучасних технологій та інструментів, що забезпечують високий рівень продуктивності, масштабованості, зручності, ефективності та безпеки. Основні технології, що використовуються, включають:

1. **Мова програмування та фреймворки:** Для розробки користувацької логіки використано мову програмування TypeScript разом із бібліотекою React, що забезпечує швидке та якісне відображення сторінок, а також надає можливість створювати великі, складні та ефективні системи на користувацької частині застосунку.

   Альтернативою могли би бути мови та фреймворки, такі як JavaScript або TypeScript з Angular, JavaScript або TypeScript з Vue. Хоча Angular та Vue є потужними та популярними фреймворками, вони не можуть

забезпечити таку ж ефективність та швидкість відображення сторінок, мають набагато складнішу логіку для створення ледачих сторінок, які відображаються лише, коли повністю зарендерилися, а також не мають в собі купи зручних і корисних інструментів, як функціональні компоненти та інші.
Щодо мови JavaScript, то вона дійсно могла би стати альтернативною, але через відсутність типізації та досить вільний стиль кодування, це могло би викликати величезну кількість помилок при роботі з даними.

Висновок: Тому вирішено використати TypeScript разом із React, так як вони забезпечує кращу ефективність, швидкість, безпеку та перелік інструментів, що покращує якість та швидкість розробки.

2. **Сховище станів:** Для зберігання поточної інформації на користувацької стороні про користувачів, їх дії, статті та інші дані отримані з серверу використовується бібліотека Redux.

   Альтернативи бібліотеці Redux для управління станами в React-додатках включають

   - MobX: Бібліотека керування станом, яка використовує спостереження за станом та реактивні обчислення. MobX автоматично оновлює стан компонентів при зміні даних.
   - Recoil: Бібліотека від Facebook, яка надає більш гнучкий та масштабований спосіб керування станом, включаючи підтримку атомів та селекторів для керування станом та похідними значеннями.
   - Zustand: Полегшене рішення для управління станами, яке використовує hooks для створення глобальних і локальних станів.
   - XState: Бібліотека для управління станами на основі кінцевих автоматів та оркестрування.

   Чому Redux може бути кращим:

   - Передбачуваність і детермінізм: У Redux стан є незмінним і управляється за допомогою чистих функцій (редукторів). Це спрощує тестування та передбачуваність поведінки додатків.

- Чітка структура та організація: Redux змушує розробників дотримуватися певної структури (дії, редуктори, сховище), що полегшує масштабування та підтримку коду у великих додатках.
- Інструменти для налагодження: Redux DevTools надають потужні інструменти для перегляду та відстеження змін стану, що полегшує налагодження складних додатків.
- Сумісність з React: Redux легко інтегрується з React через react-redux, що дозволяє ефективно прив'язувати стан до компонентів і використовувати хуки, такі як useSelector і useDispatch.

Висновок: Використання бібліотеки Redux забезпечує безпечність, зручність і структурованість при роботі зі станами в React а також вона є дуже потужним інструментом для складних додатків, при цьому не вимагає значних витрат часу на конфігурацію та подібне!

3. **Маршрутизація:** Для впровадження логіки перемикання між сторінками була використана бібліотека wouter.

   Альтернативи бібліотеці Wouter для керування маршрутизацією в React-додатках включають:

   - React Router: Найпопулярніша бібліотека маршрутизації в React, що надає повний набір функцій для роботи з динамічними маршрутами, вкладеними маршрутами та параметрами URL.

   - Reach Router: Легка бібліотека маршрутизації, призначена для створення доступних та адаптивних веб-додатків. Має простий та інтуїтивно зрозумілий API.

   - Next.js: Фреймворк для React, який підтримує маршрутизацію на основі файлової системи, що дозволяє легко створювати додатки з наданням послуг та статичні сайти.

   - Routify: Бібліотека для маршрутизації, яка дозволяє працювати з компонентами та забезпечує підтримку гнучкого налаштування маршрутів.

   Чому Wouter краще:

- Легкий і простий: Wouter - це легка бібліотека, яка додає мінімальний розмір до пакета програми, що покращує продуктивність. Її API простий та інтуїтивно зрозумілий, що робить її легкою у налаштуванні та використанні.
- Компонентний підхід: Wouter використовує хуки та компоненти для маршрутизації, що дозволяє легко інтегрувати його в сучасні React-додатки, написані за допомогою функціональних компонентів.
- Гнучкість: Wouter надає можливість визначати маршрути без додаткових складнощів і дозволяє легко реалізовувати динамічні маршрути, обробляти параметри URL та інші аспекти маршрутизації.
- Відсутність залежностей: Wouter не має зовнішніх залежностей, що знижує ризик конфліктів і забезпечує стабільність роботи програми.
- Сумісність з React: Wouter легко інтегрується з React і підтримує такі хуки, як useLocation, useRoute і useNavigate, що дозволяє ефективно працювати з маршрутизацією у вашому додатку.

Висновок: Використання бібліотеки Wouter забезпечує легкість, зручність та простоту при реалізації логіки перемикання сторінок у React-додатках. Це потужний інструмент для додатків, які потребують простої та ефективної маршрутизації без додаткової складності та зайвої ваги, що дуже підходить сайту Студабр.

4. **Фреймворк для дизайну:** Для реалізації стилів та компонентів у додатку було використано фреймворк Bootstrap.

   Існують альтернативи фреймворку Bootstrap для стилізації та побудови компонентів у React-додатках:

   - Material-UI (MUI): Популярний фреймворк React, заснований на концепціях Material Design від Google. Надає велику кількість готових компонентів та можливостей кастомізації
   - Ant Design: Фреймворк з великим набором компонентів і потужними можливостями кастомізації, популярний у бізнес-додатках і корпоративних рішеннях.

- Chakra UI: Модульний і доступний фреймворк для React, який надає безліч готових компонентів і простий у використанні API для налаштування стилів.
- Tailwind CSS: Утиліта CSS-фреймворк, що дозволяє створювати дизайни без написання власних класів CSS. Забезпечує високу гнучкість і швидкість розробки.
- Bulma: Сучасний CSS-фреймворк, який надає прості та гнучкі компоненти, простий у вивченні та використанні.

Чому Bootstrap краще?

- Широка популярність і спільнота: Bootstrap - один з найпопулярніших фреймворків для стилізації веб-додатків. Це забезпечує велику кількість ресурсів, навчальних посібників та підтримку спільноти.
- Готові компоненти: Bootstrap надає великий набір готових компонентів, таких як кнопки, форми, навігаційні панелі, модальні вікна тощо. Це значно прискорює розробку і дозволяє зосередитися на логіці роботи додатка.
- Адаптивний дизайн: Bootstrap має вбудовану підтримку адаптивного дизайну, що дозволяє легко створювати додатки, які добре виглядають на різних пристроях.
- Простота використання: Завдяки простому і зрозумілому синтаксису, Bootstrap легко вивчити і використовувати навіть новачкам у веб-розробці.
- Сумісність з React: Існують бібліотеки, такі як react-bootstrap, які дозволяють легко інтегрувати Bootstrap-компоненти в React-додатки за допомогою JSX.

Висновок: Використання фреймворку Bootstrap забезпечує швидкий старт, простоту використання та велику кількість готових рішень для стилізації та компонентів у веб-додатках. Це робить його потужним інструментом для створення красивих і адаптивних інтерфейсів.

5. **Текстовий редактор:** Для створення текстового редактора у додатку була використана бібліотека Lexical.

   Альтернативи бібліотеці Lexical для створення текстових редакторів у React-додатках включають:

- Draft.js: Бібліотека для створення редакторів у React від Facebook. Забезпечує високу гнучкість та розширюваність за допомогою плагінів.
- Slate: Потужна бібліотека для створення текстових редакторів, що дозволяє повністю контролювати структуру документа та поведінку редактора.
- Quill: Легка і проста у використанні бібліотека текстових редакторів, яка надає широкий спектр можливостей та інтуїтивно зрозумілий API.
- TinyMCE: Популярний WYSIWYG-редактор, який надає потужні можливості редагування тексту та великий набір інструментів для налаштування.
- CKEditor: Високофункціональний WYSIWYG-редактор з великою кількістю плагінів і можливостями для інтеграції з різними платформами.

Чому Lexical краще?

- Продуктивність: Lexical розроблено з урахуванням продуктивності, що робить його швидким і ефективним при роботі з великими текстами та складними структурами документів.
- Гнучкість: Lexical дозволяє розробникам легко створювати та налаштовувати текстові редактори під конкретні потреби додатків, забезпечуючи високу гнучкість і розширюваність.
- Сучасний підхід: Lexical використовує сучасні методи та технології, що дозволяє легко інтегруватися з іншими сучасними інструментами та фреймворками.
- Підтримка React: Lexical добре інтегрується з React, надаючи простий і зрозумілий API для створення текстових редакторів у React-додатках.

Висновок: Lexical надає безліч зручних та простих інструментів для швидкої та ефективної розробки текстового редактора. Вона дуже гнучка та продуктивна, має типізацію та надає можливість, як налаштовувати існуючу, так і створювати нову логіку за допомогою наданих інтсрументів, це робить її найкращим варіантом для використання на сайті Студабр.

6. **Бібліотека для HTTP-запитів:** Додаток використовує бібліотеку Axios для створення HTTP-запитів.

   Альтернативи бібліотеці Axios для створення HTTP-запитів у React-додатках включають:

   - Fetch API: Вбудований у браузер API для створення HTTP-запитів. Простий у використанні і не вимагає додаткових залежностей.
   - Суперагент: Гнучка бібліотека для виконання HTTP-запитів з підтримкою обіцянок та великою кількістю можливостей кастомізації.
   - axios-hooks: Бібліотека для використання Axios як хуків React, що полегшує роботу з HTTP-запитами у функціональних компонентах.
   - SWR: бібліотека від Vercel для роботи з даними, яка використовує філософію stale-while-revalidate для ефективного управління кешем та повторного отримання даних.
   - React Query: Потужна бібліотека для керування станом серверних даних у React-додатках, що забезпечує ефективне керування кешем, синхронізацію даних та повторні запити.

   Чому був обраний Axios:

   - Простота використання: Axios має інтуїтивно зрозумілий API, що робить його простим у використанні для виконання HTTP-запитів. Він підтримує синтаксис на основі обіцянок, що дозволяє легко працювати з асинхронними запитами.
   - Широка функціональність: Axios надає багато корисних функцій, таких як перехоплювачі запитів і відповідей, автоматичне перетворення JSON, тайм-аути, обробка помилок і підтримка скасування запитів.
   - Конфігурація: Axios дозволяє легко налаштовувати параметри запиту, такі як заголовки, базову URL-адресу, параметри виконання тощо. Це робить його дуже гнучким і простим у використанні в різних проектах.
   - Сумісність з браузерами та Node.js: Axios працює як в браузерах, так і в середовищі Node.js, що дозволяє використовувати його як для front-end, так і для back-end розробки.

- Підтримка перехоплювачів: Axios надає можливість використовувати перехоплювачі для обробки або модифікації запитів і відповідей, що значно розширює його функціональність.

Висновок: Використання бібліотеки Axios забезпечує простоту, зручність та потужну функціональність для виконання HTTP-запитів у React-додатках. Вона пропонує широкий набір інструментів для кастомізації та обробки запитів, що робить її чудовим вибором для реалізації складних та гнучких рішень у веб-додатку Студабр.

7. **Збірка:** Для збірки коду у додатку був використаний Webpack.

   Альтернативи Webpack для збірки JavaScript-додатків включають:

   - Parcel: Інструмент для збірки з мінімальними налаштуваннями, який автоматично налаштовує багато аспектів збірки і має вбудовану підтримку сучасних функцій.
   - Rollup: Інструмент для збірки, оптимізований для бібліотек, який забезпечує маленькі розміри бандлів і підтримує tree-shaking для видалення невикористаного коду.
   - Vite: Інструмент для збірки від творців Vue.js, який забезпечує швидкий старт і гаряче перезавантаження модулів завдяки використанню нативних ES-модулів.
   - Snowpack: Інструмент для збірки, який дозволяє використовувати сучасні браузерні функції для швидкої розробки з нативними ES-модулями.

   Чому був обраний Webpack:

   - Гнучкість: Webpack є надзвичайно гнучким і настроюваним інструментом, який дозволяє налаштувати практично всі аспекти збірки через конфігураційні файли.
   - Широка екосистема: Webpack має величезну кількість плагінів і лоудерів, які розширюють його можливості і дозволяють працювати з різними типами файлів і форматів.
   - Оптимізація продуктивності: Webpack пропонує потужні інструменти для оптимізації бандлів, такі як code splitting, lazy loading і tree-shaking, що дозволяє зменшити розмір і покращити продуктивність додатків.

- Сумісність з різними фреймворками: Webpack добре інтегрується з різними JavaScript-фреймворками і бібліотеками, такими як React, Vue, Angular та інші.
- Підтримка гарячого перезавантаження: Webpack DevServer підтримує гаряче перезавантаження модулів, що покращує продуктивність розробки і скорочує час на оновлення додатка під час змін у коді.

Висновок: Використання Webpack забезпечує гнучкість, потужні можливості для налаштування і оптимізації, що робить його ідеальним інструментом для збірки складних JavaScript/TypeScript додатків. Його широка екосистема і підтримка різних фреймворків дозволяють легко адаптувати Webpack під будь-які потреби проекта, а також зберігає купу часу при конфігурації проекту, що робить цей інструмент незамінним для сайту Студабр.

8. **Транспіляція:** Для транспіляції коду у додатку був використаний Babel.

   Альтернативи Babel для транспіляції JavaScript-коду включають:

   - SWC (Speedy Web Compiler): Швидкий транспілятор і мінімізатор для JavaScript і TypeScript, який забезпечує високу продуктивність завдяки використанню Rust.
   - Esbuild: Дуже швидкий транспілятор і бандлер для JavaScript і TypeScript, який також написаний на Go і відомий своєю швидкістю.
   - TypeScript Compiler (tsc): Використовується переважно для транспіляції TypeScript у JavaScript, але також може працювати з сучасними функціями JavaScript.

   Чому був обран Babel:

   - Підтримка нових стандартів: Babel швидко додає підтримку нових функцій JavaScript, що дозволяє розробникам використовувати найновіші можливості мови, незалежно від підтримки браузерами.
   - Екосистема плагінів: Babel має велику кількість плагінів, які дозволяють налаштувати транспіляцію під конкретні потреби, включаючи підтримку JSX, TypeScript і багатьох інших синтаксичних розширень.

- Гнучкість налаштувань: Babel дозволяє налаштувати процес транспіляції за допомогою конфігураційних файлів, що дає можливість адаптувати його під проект.
- Сумісність з іншими інструментами: Babel легко інтегрується з Webpack, Rollup, Parcel та іншими інструментами для збірки, що робить його універсальним рішенням для транспіляції.
- Підтримка полілфіллів: Babel може автоматично додавати полілфілли для нових функцій JavaScript, що забезпечує крос-браузерну сумісність.

Висновок: Використання Babel забезпечує гнучкість і потужні можливості для транспіляції сучасного JavaScript-коду у більш сумісний формат. Це робить його важливим інструментом для використання новітніх можливостей мови, при цьому він забезпечує сумісність з усіма сучасними браузерами та середовищами виконання. Це все робить Babel дуже гарним вибором для сайту Студабр.

1.3. Аналіз функціоналу, який потрібно розробити

Для забезпечення повноцінного функціонування сайту Студабр було розроблено наступний функціонал:

1. Реєстрація та аутентифікація користувачів: Створені логіка для реєстрації нових користувачів та подальшого входу в систему, а також сторінки та компоненти, з якими працює користувач.
2. Створення публікацій: Створені сторінки, компоненти з якими працює користувач, а також логіка для обробки, зберігання, валідації та передання даних на сервер.
3. Створення текстового редактора: Створені компоненти, вузли, методи для текстового редактора. Створена власна бібліотека, яка додає можливість додавати картинки двома способами: завантажити с файлової системи або вказати посилання на неї. Створені модулі відповідальні за форматування тексту, а також панель інструментів.
4. Уподобайки: Реалізован компонент для постів, щоб забезпечити взаємодію з публікаціями через уподобайки. Створена логіка для запиту на оновлення списку постів з уподобайками.

5. Перегляд публікацій: Створенні компоненти та сторінки для перегляду як окремих публікацій, так і їх списку, а також логіка для пошуку потрібної публікації та пагінація.

Цей функціонал забезпечує виконання основних задач сайту Студабр.

2. ПРОЕКТУВАННЯ КОРИСТУВАЦЬКОЇ ЧАСТИНИ САЙТУ СТУДАБР

2.1. Архітектурні рішення та структура системи

Користувацька частина сайту Студабр організована за компонентно-орієнтованою архітектурою, що забезпечує модульність, повторне використання компонентів та легкість у підтримці. Основні компоненти системи включають:

- App (Головний компонент): Відповідає за загальну структуру застосунку та маршрутизацію. Імпортує глобальні стилі та налаштовує контексти.
- Pages (Сторінки): Містять компоненти, що відповідають за рендеринг окремих сторінок сайту. Наприклад, Welcome, Article, Profile, Login, Register.
- Components (Компоненти): Переважно повторно використовувані елементи інтерфейсу, такі як кнопки, форми, модальні вікна тощо. Кожен компонент організований в окремій теці, що включає файл компонента, стилі та тести.
- Services (Сервіси): Містять логіку для взаємодії з API. Наприклад, AuthService для аутентифікації користувачів, UserService для роботи з профілями користувачів, ArticleService для управління статтями.
- Hooks (Хуки): Користувацькі React-хуки, що абстрагують логіку взаємодії з сервісами та контекстами. Наприклад, useAuth, useArticles, useUser.
- Utils (Утиліти): Містять допоміжні функції та утилітарні модулі. Наприклад, функції для форматування дат, обробки помилок, валідації форм.
- Assets (Ресурси): Статичні файли, такі як зображення, іконки, шрифти та інші медіафайли.

Приклад структури папок та файлів:

```
src/
├── components/
│   ├── Button/
│   │   ├── Button.tsx
│   │   ├── Button.styles.ts
│   │   └── Button.test.tsx
│   ├── Header/
│   │   ├── Header.tsx
│   │   ├── Header.styles.ts
│   │   └── Header.test.tsx
│   └── ...
├── pages/
│   ├── Home/
│   │   ├── Home.tsx
│   │   ├── Home.styles.ts
│   │   └── Home.test.tsx
│   ├── Article/
│   │   ├── Article.tsx
│   │   ├── Article.styles.ts
│   │   └── Article.test.tsx
│   └── ...
├── App.tsx
├── index.tsx
```

Use Case діаграма:

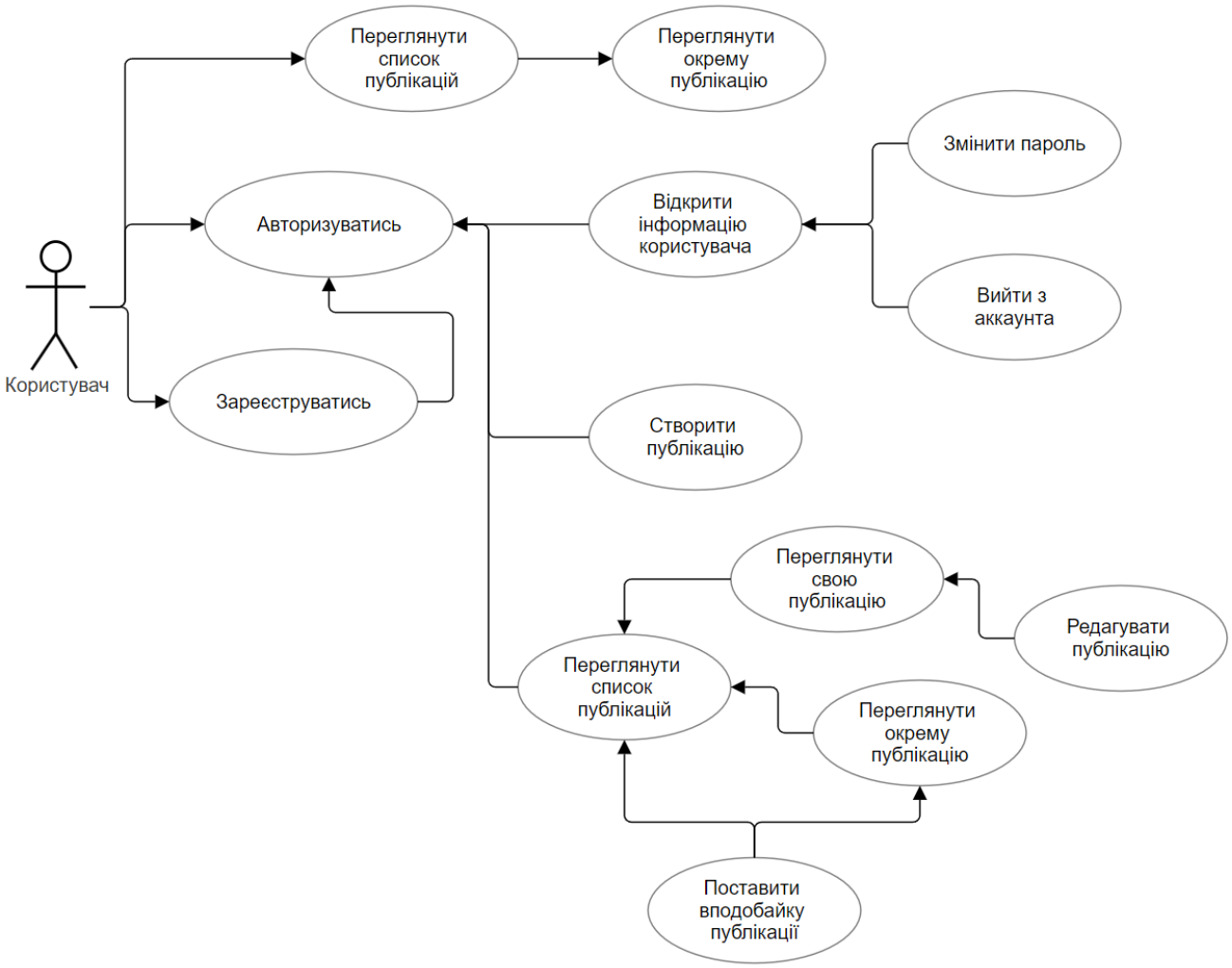


Рис. 1. Use Case діаграма

2.2. Опис та конфігурація сховища станів

Для зберігання станів використовується Redux, що забезпечує гнучкість у роботі з даними і високу ефективність. Основні моделі та їх структура:

- Користувачі (Users):

  Схема:

```
export interface User {

  slug: string;

  username: string;
```

```
  email: string;

  bio: string;

  image: string;

  password: string;

}
```

- Публікації(Articles):

```
export interface Blog {
  title: string;
  slug: string;
  subtitle: string;
  createdDate: string;
  author: any;
  content: string;
  mainImageLink: string;
  tagList: string[];
  favoritesCount: number;
}
```

У цьому стані зберігається інформація про статті, включаючи їх авторів та кількість вподобань.

- Фоловери (Followers):

```
export interface Follow {

  follower: string;

  following: string;

}
```

Для кожної моделі були створені редуктори, які відповідають за оновлення станів. Загалом вони поділяються на 3 види:

- Редуктор ініціалізації оновлення стану - наприклад, відправлення HTTP запиту на сервер, для отримання даних. Ми повідомляємо Redux про це через редуктор, щоб стан був готовий до оновлення.
- Редуктор успішного оновлення стану - використовується якщо ми успішно отримали потрібні дані та в такому разі зберігаємо їх в стані.
- Редуктор помилок - використовується в разі отримання помилки від серверу або проблеми з коректністю даних. Він ініціалізує поле для помилки в стані і заповнює його.

Приклад редуктору з використанням createSlice:

```
import { PayloadAction, createSlice } from
'@reduxjs/toolkit';
import { Blog } from '../../models/Blog';

interface BlogState {
  blog: Blog[];
  isLoading: boolean;
  error: string;
}

const initialState: BlogState = {
  blog: [],
  isLoading: false,
  error: '',
};

export const blogSlice = createSlice({
  name: 'user',
  initialState,
  reducers: {
    getblogStart: (state) => {
      state.isLoading = true;
```

```
    },
    getblogSuccess: (state, action: PayloadAction<Blog[]>) =>
{
      state.blog = action.payload;
      state.isLoading = false;
    },
    getblogFailure: (state, action: PayloadAction<string>) =>
{
      state.error = action.payload;
      state.isLoading = false;
    },
  },
});

export default blogSlice.reducer;
```

Далі нам потрібно вказати наші редуктори в конфігурації Redux та додати типізацію:

```
import { combineReducers, configureStore } from
'@reduxjs/toolkit';
import userReducer from './reducers/UserSlice';
import blogReducer from './reducers/BlogSlice';
import postReducer from './reducers/PostSlice';

const rootReducer = combineReducers({
  userReducer,
  blogReducer,
  postReducer,
  favoritesReducer
});

export const setupStore = () => {
  return configureStore({
    reducer: rootReducer,
```

```
  });
};

export type RootState = ReturnType<typeof rootReducer>;
export type AppStore = ReturnType<typeof setupStore>;
export type AppDispatch = ReturnType<typeof
setupStore>['dispatch'];Hi
```

UML Діаграма Активностей:

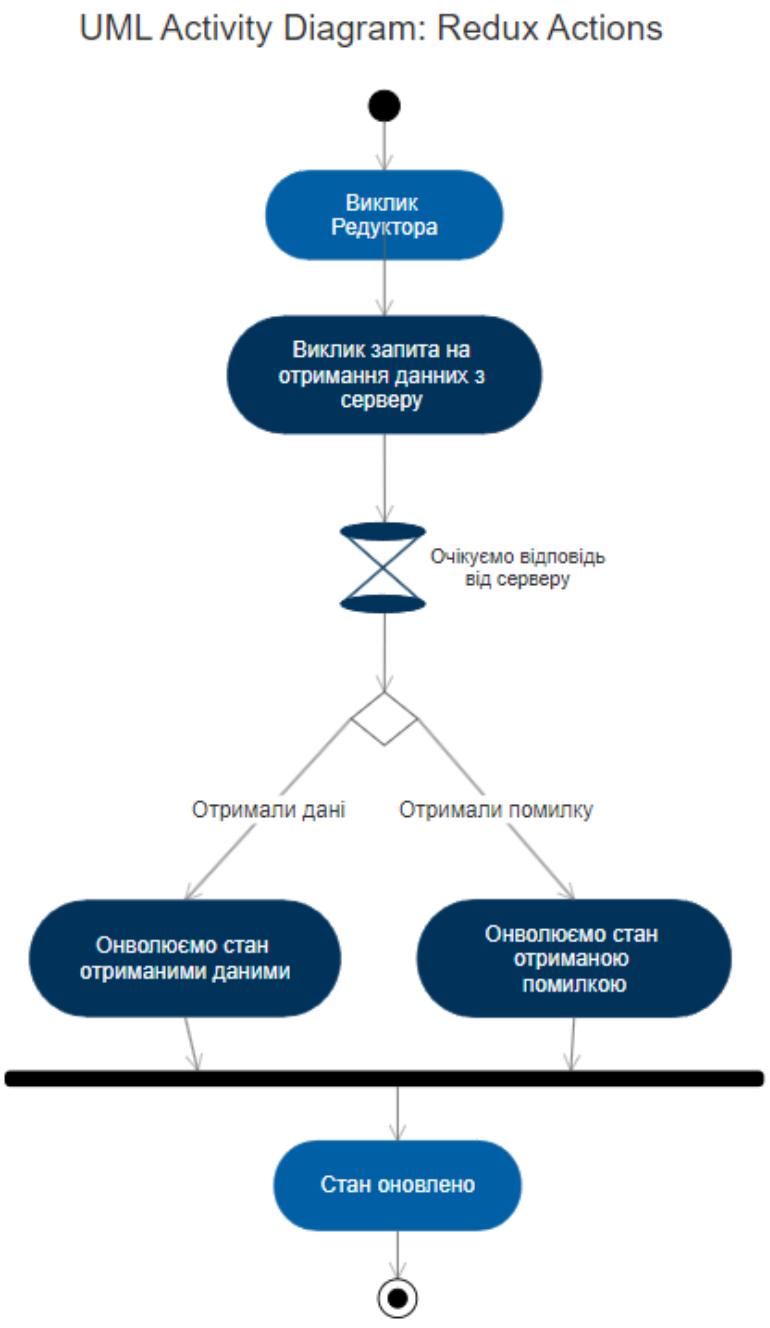


Рис. 2. Діаграма активностей Redux сховища

2.3. Опис конфігурації бібліотеки Axios

Цей розділ описує конфігурацію Axios для взаємодії з серверною частиною нашого веб-застосунка. Використовуючи цю конфігурацію, ми забезпечуємо належне управління JWT-токеном, який зберігається в куках, та обробку помилок запитів.

Конфігурація:

- Налаштування інтерсепторів запитів: Інтерсептори запитів дозволяють перехоплювати запити перед їх відправленням на сервер. У нашому випадку, ми додаємо JWT-токен до заголовків кожного запиту, якщо він доступний у куках.

- Налаштування інтерсепторів відповідей: Інтерсептори відповідей дозволяють обробляти помилки. Якщо сервер повертає помилку 401 (Unauthorized), ми видаляємо токен з куків та перенаправляємо користувача на сторінку входу.

2.4. Опис сервісів для взаємодії з API

Сервіси для взаємодії з API були реалізовані за допомогою бібліотеки Axios, яка дає зручні та ефективні інструменти для HTTP-запитів.

Основні сервіси та їх методи:

- Сервіс публікацій:
    - fetchPublications - виконує запит GET /articles/get-all для отримання всіх постів.
    - fetchPublication - виконує запит GET /articles/get-single/:slug для отримання публікації, у якості параметра передає slug публікації.
    - createPublication - виконує запит POST /articles/create, та передає в якості тіла запиту поля для нової публікації.
    - deletePublication - виконує запит DELETE /articles/delete/:slug для видалення публікації, у якості параметра передає slug публікації.

    - updatePublication - виконує запит PUT /articles/update/:slug для оновлення публікації, в тілі запиту передає нові поля для неї.
    - fetchPublicationsWithSearch - виконує запит GET /articles/search-by-keyword для пошуку публікацій за пошуковим запитом користувача

- Сервіс користувача:
    - login - виконує запит POST /user/login для авторизації користувача у додатку, у якості тіла запиту передає логін та пароль.
    - register - виконує запит POST /user/create для реєстрації нового користувача, у якості тіла запиту передає введені користувачем дані.
    - fetchCurUser - виконує запит GET /user/current-user для отримання даних про поточного користувача.
    - updateCurUser - виконує запит PUT /user/update-current для оновлення даних поточного користувача.
    - fetchUser - виконує запит GET /user/get-single/:slug для отримання даних користувача, в якості параметра передає slug користувача.

- Сервіс уподобайок:
    - favorite - виконує запит POST /articles/:slug/favorite для додавання публікації до тих що сподобались.
    - unfavorite - виконує запит DELETE /articles/:slug/unfavorite для видалення публікації з тих що сподобались.

Для забезпечення безпеки, зі сторони клієнта, до кожного запиту прикріпляється також JWT-token, якщо він є в куках, а також валідуються дані перед відправленням.

3. РОЗРОБКА КОРИСТУВАЦЬКОЇ ЧАСТИНИ САЙТУ СТУДАБР

3.1. Вибір та налаштування середовища розробки

1. Середовище розробки:
    - Використовуване середовище: Visual Studio Code
    - Плагіни та розширення для полегшення роботи з кодом:
        - Auto Close Tag: відповідає за автоматичне закриття HTML та XML тегів.
        - Auto Rename Tag: відповідає за автоматичне перейменування закриваючого HTML та XML тегу, коли відкриваючий змінюється.
        - ESLint: відповідає за виявлення та виправлення логічних та синтаксичних помилок у JavaScript/TypeScript або JSX коді. Також відповідає за форматування коду.
        - Prettier - Code formatter: відповідає за автоматичне форматування коду для забезпечення консистентності.
        - WSL: відповідає за підтримку Windows Subsystem for Linux для розробки у Linux середовищі на Windows.
        - GitHub Copilot: є асистентом, допомагає при написанні коду, прискорює процес розробки.
        - Auto Import: знаходить та підказує можливі шляхи до файлів при імпортуванні функціонала, компонентів тощо.
        - Import Cost: відповідає за відображення розміру імпорту для кращого використання пам’яті.
        - HTML CSS Support: відповідає за підказки при написанні HTML тегів, CSS коду та класів.
        - Markdown Lint: відповідає за дотримання відступів та єдиного стилю написання для маркдаун файлів.
2. Мова програмування та інструменти:
    - Основна мова програмування: TypeScript
    - Інструменти та бібліотеки:
        - React: це користувацька JavaScript/TypeScript бібліотека для створення інтерфейсів користувача (UI). React дозволяє розробникам будувати веб-додатки, використовуючи компонентний підхід, що полегшує управління і повторне використання коду.
        - Axios: TypeScript/JavaScript бібліотека для виконання HTTP-запитів з браузера або Node.js, що підтримує

проміси, інтерцептори запитів та відповідей, та автоматичну трансформацію даних. TypeScript типи доступні з коробки.

- Lexical: TypeScript/JavaScript бібліотека для створення текстових редакторів, що забезпечує високу продуктивність, модульність та розширюваність, дозволяючи легко створювати і налаштовувати редактори. TypeScript підтримується для типізації редактора та його плагінів.
- Redux: TypeScript/JavaScript бібліотека для управління станом додатка, яка дозволяє централізувати стан та логіку бізнесу, підтримуючи передбачуваність та відлагоджуваність коду.
- Bootstrap/React-Bootstrap: CSS-фреймворк та набір React-компонентів для швидкої та легкої розробки адаптивних та сучасних інтерфейсів користувача. React-Bootstrap надає типи для TypeScript з коробки, що полегшує роботу з компонентами в TypeScript-додатках.
- Styled-Components: бібліотека для CSS-in-JS, яка дозволяє писати стилі у файлах JavaScript/TypeScript. Вона використовує шаблонні літерали для створення компонентів зі стилями.
- Mocha: це тестовий фреймворк для Node.js та браузера, що дозволяє писати асинхронні тести та забезпечує гнучкість при виборі бібліотек для тверджень (assertion).
- Chai: це бібліотека для тверджень (assertion), яка може використовуватися з будь-яким тестовим фреймворком, включаючи Mocha.
- @testing-library/react: це бібліотека для тестування React-компонентів, вона забезпечує легке та інтуїтивне API для взаємодії з компонентами через їх рендеринг, знаходження елементів та обробляє різні дії, наприклад натискання кнопок та інше.
- CLSX: це невелика утиліта для умовного об'єднання класів CSS, особливо корисна для управління класами в React-компонентах.

- JS-cookie: бібліотека для роботи з кукі в JavaScript, яка забезпечує простий API для встановлення, отримання та видалення кукі.
- React-dom: це бібліотека, яка надає специфічні для DOM методи, що дозволяють використовувати React для рендерингу на веб-сторінці.
- React-icons: це бібліотека, що надає колекцію іконок для використання в React-додатках.
- Wouter: це маленька, але потужна бібліотека для маршрутизації в React, альтернативна React Router, що зосереджена на швидкості та простоті.
- React-scripts: це пакет, що надається Create React App для налаштування та запуску React-додатків без необхідності конфігурації.
- React-test-renderer: це бібліотека для рендерингу React-компонентів в чистий JavaScript об'єкт для цілей тестування.
- Prettier: це інструмент для форматування коду, що забезпечує консистентність стилю коду у проектах, незалежно від того, хто його пише.
- SASS: це препроцесор для CSS, що додає додаткові можливості, такі як змінні, вкладеність, міксини та інше.
- TypeScript: це надбудова над JavaScript, що додає статичну типізацію. Він допомагає зменшити кількість помилок під час розробки та робить код більш зрозумілим і підтримуваним.

3. Система контролю версій:

Система контролю версій: Для управління версіями коду у проекті була використана система контролю версій Git.

Альтернативи системі контролю версій Git включають:

- Subversion (SVN): Централізована система контролю версій, яка забезпечує можливість роботи над проектом з одного репозиторію.

- Mercurial: Децентралізована система контролю версій, яка надає схожий до Git функціонал, але з іншим підходом до управління репозиторіями.
- Perforce Helix Core: Система контролю версій для великих проектів, яка забезпечує високу продуктивність і масштабованість.
- Bazaar: Інструмент для децентралізованого управління версіями, має акцент на простоту використання і інтеграцію з іншими інструментами.
- Fossil: Система контролю версій, яка включає багатофункціональний набір інструментів для управління проектами, також дає можливість відстежувати помилки та створювати вікі.

Чому саме Git:

- Децентралізація: Git є розподіленою системою контролю версій, це дозволяє кожному розробнику мати повну копію репозиторію, що забезпечує високу надійність та гнучкість.
- Потужні можливості для гілкування та злиття: Git надає потужні інструменти для роботи з гілками і змінами в коді, що дозволяє ефективно керувати розробкою і інтеграцією нових функцій.
- Інтеграція з хмарними сервісами: Git легко інтегрується з популярними хмарними сервісами для керування репозиторіями, такими як GitHub, GitLab і Bitbucket, що спрощує управління проектами.
- Висока продуктивність: Git дуже швидко працює, навіть з великими репозиторіями і великою кількістю змін, що робить його дуже ефективним інструментом
- Гнучкість у налаштуванні: Git надає багато можливостей для налаштування і автоматизації процесів, це дозволяє налаштовувати автоматичні перевірки коду.

Висновок: Використання системи контролю версій Git забезпечує децентралізацію, потужні можливості для гілкування і злиття, високу продуктивність і широкі можливості для інтеграції з іншими інструментами та сервісами. Це робить Git незамінним інструментом сайту Студабр.

4. Налаштування локального середовища:

- Інструкції з налаштування локального середовища:
  - Встановити pnpm з офіційного сайту.
  - Клонувати репозиторій з GitHub: git clone <URL>
  - Перейти в директорію проекту: cd <project-directory>
  - Встановити необхідні бібліотеки: pnpm install
  - Запустити додаток: pnpm start

3.2. Реалізація основних сторінок та модулів

Сторінки сайту Студабр:

- Вітальна(Welcome): Ця сторінка - це перше що бачитькористувач, коли заходить на Студабр. Тож завдання полягало в том, щоб зробити привабливий дизайн який чіпляє користувача, а також додати трохи інформації про платформу, з якою людина може ознайомитись та зацікавитися ще більше. Сторінка розділена на 3 частини:
  - Вітаємо на нашому сайті:

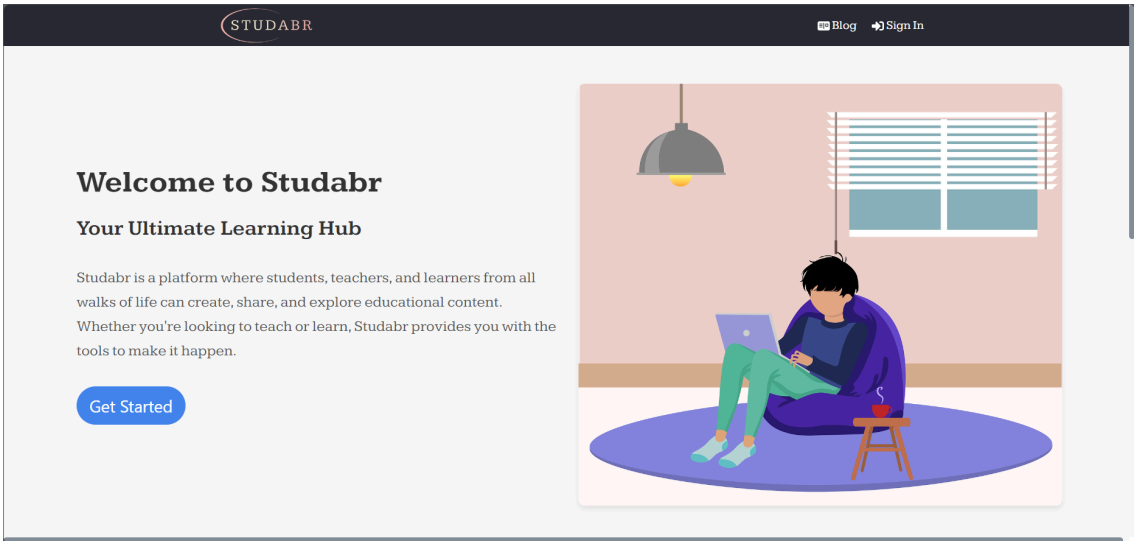


Рис. 3. Перша частина “Вітальної сторінки

- Наші переваги:

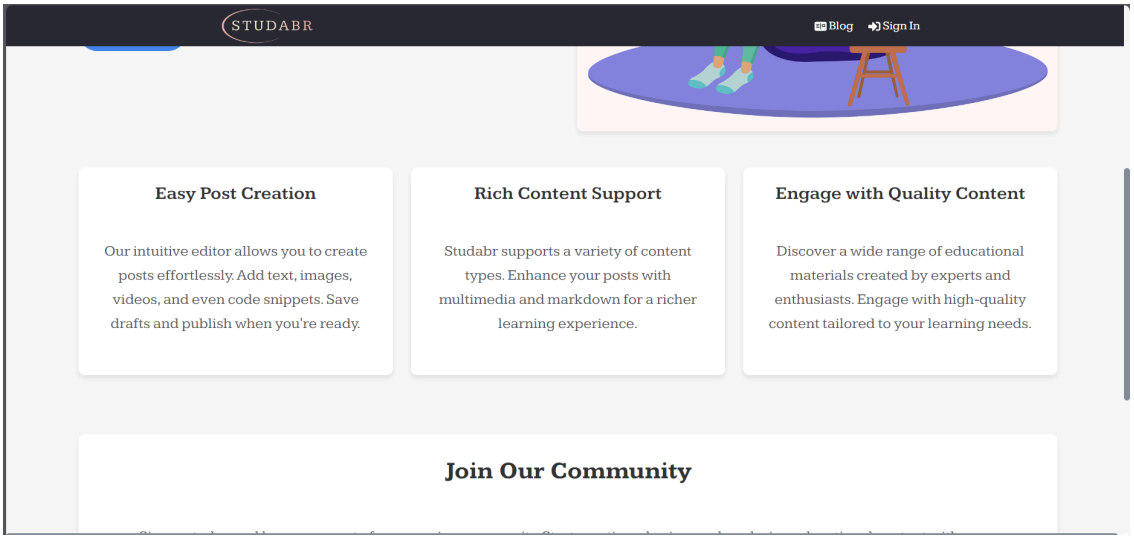


Рис. 4. Друга частина “Вітальної” сторінки

- Приєднуйтесь до нас:

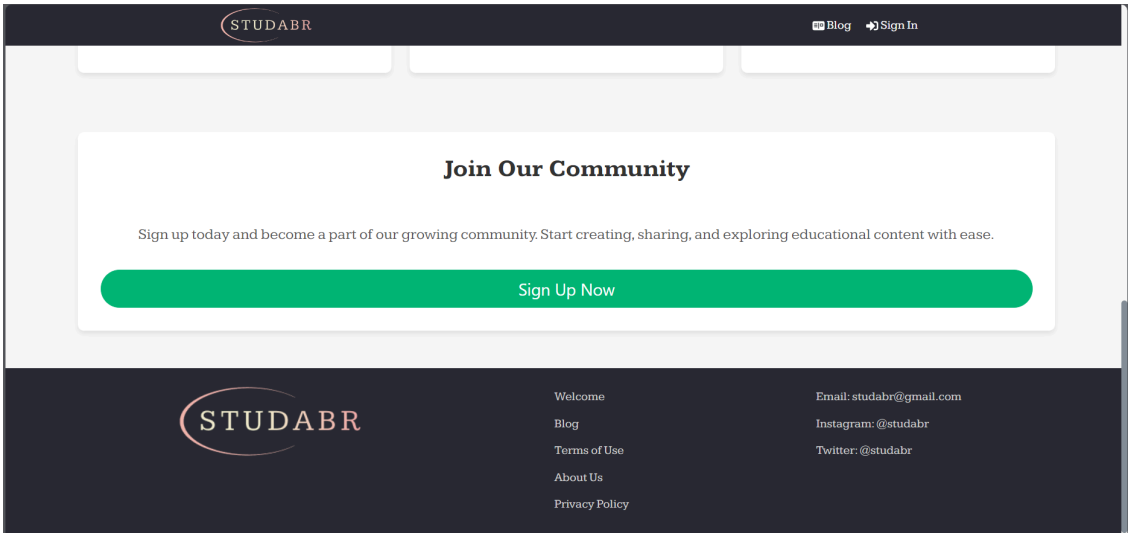


Рис. 5. Третя частина “Вітальної” сторінки

Ця сторінка є малофункціональною, вона має пару кнопок які перенаправляють користувача на інші сторінки, це

- Get Started - переносить користувача на сторінку зі списком публікацій
- Sign Up Now - переносить користувача на сторінку реєстрації

- Про нас(About Us) - Ця сторінка була створена на випадок, якщо користувач зацікавиться історією сайту, командою, головною ціллю платформи тощо. Сторінка не має в собі якоїсь

функціональності та створена лише для надання інформації
Про нас - має декілька частин:

- Вітальна частина:

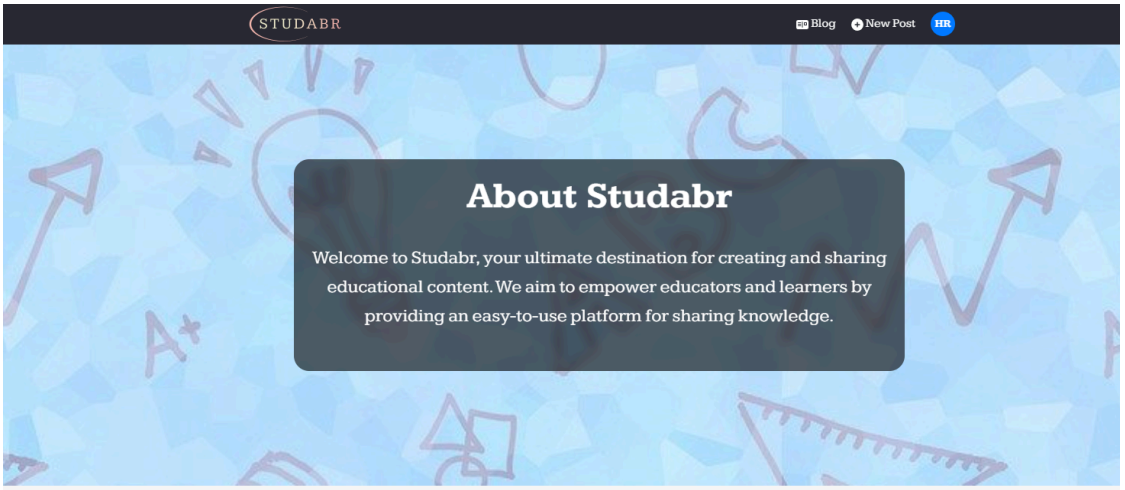


Рис. 6. Перша частина сторінки “Про Нас”

- Наша місія:

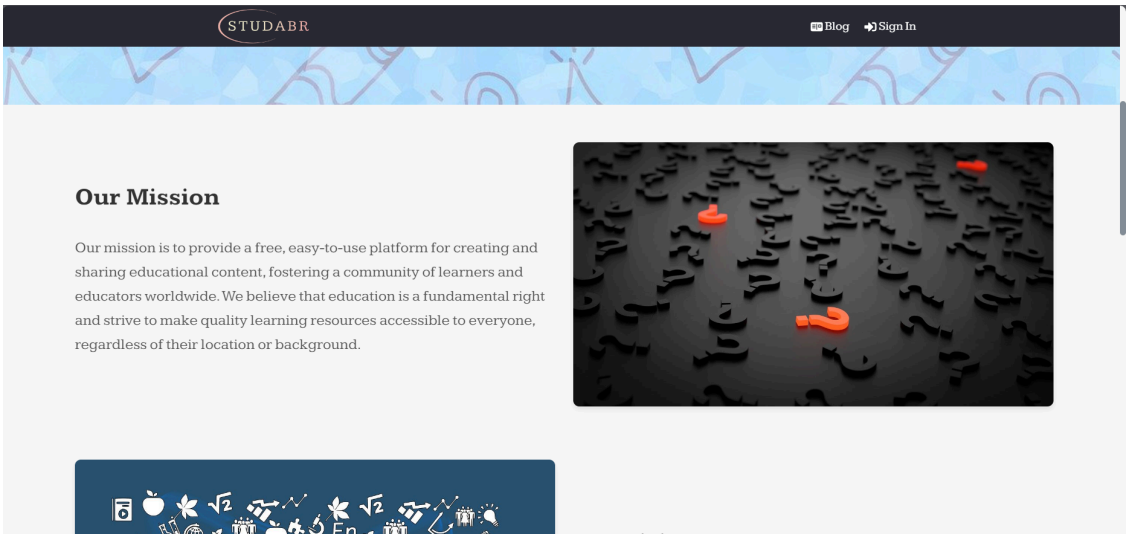


Рис. 7. Друга частина сторінки “Про Нас”

- Наше бачення:

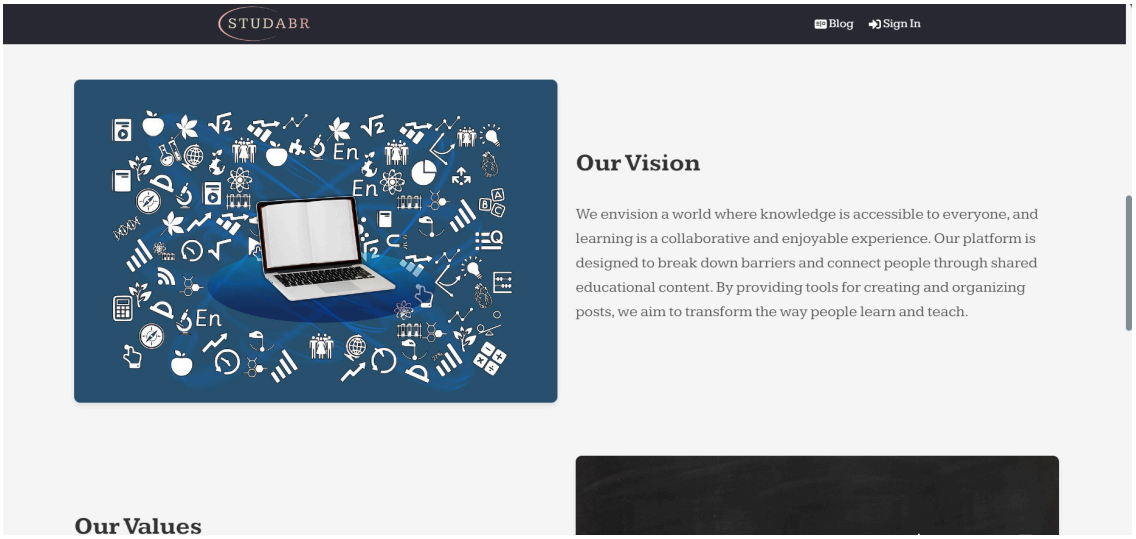


Рис. 8. Третя частина сторінки “Про Нас”

- Наші цінності:

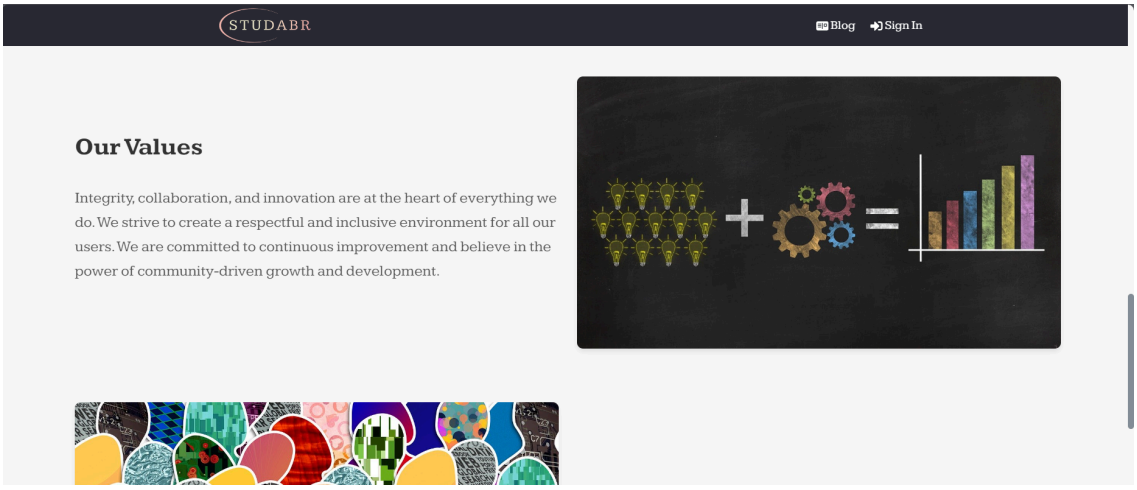


Рис. 9. Четверта частина сторінки “Про Нас”

- Наша команда:

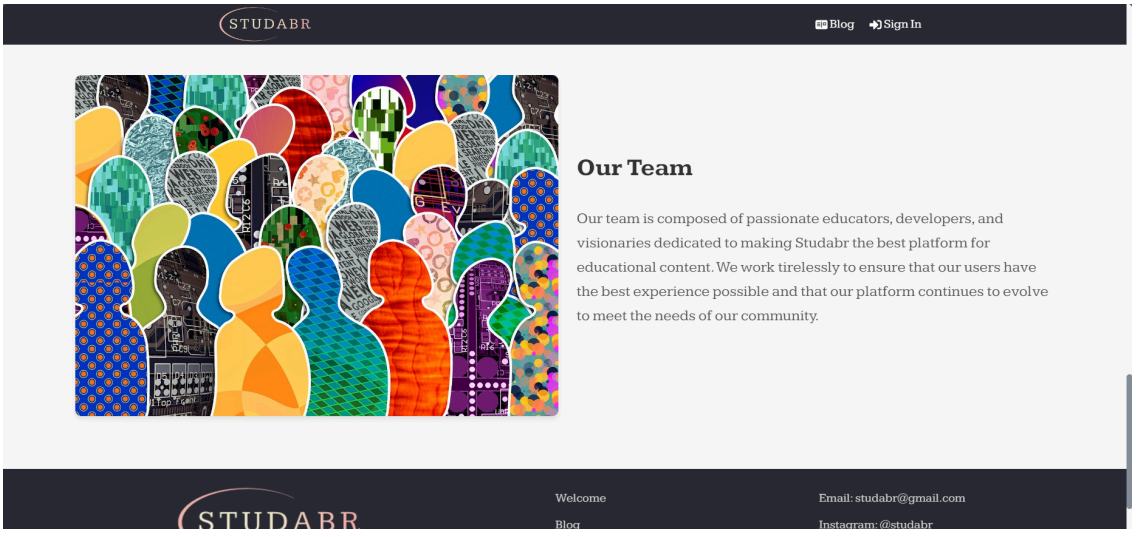


Рис. 10. П’ята частина сторінки “Про Нас”

- Умови використання(Terms Of Use) - Ця сторінка була створена для чіткого описання правових зобов'язань, правил, обмежень, правил використання інтелектуальної власності та інших юридичних питань. Було проведене невелике дослідження, щодо основних юридичних положень для веб-застосунків та грунтуючись на цих положеннях були описані умови користування веб-застосунком, які знімають юридичну відповідальність з мене, як з розробника у рамках дійсного законодавства.

    - Перша частина сторінки:

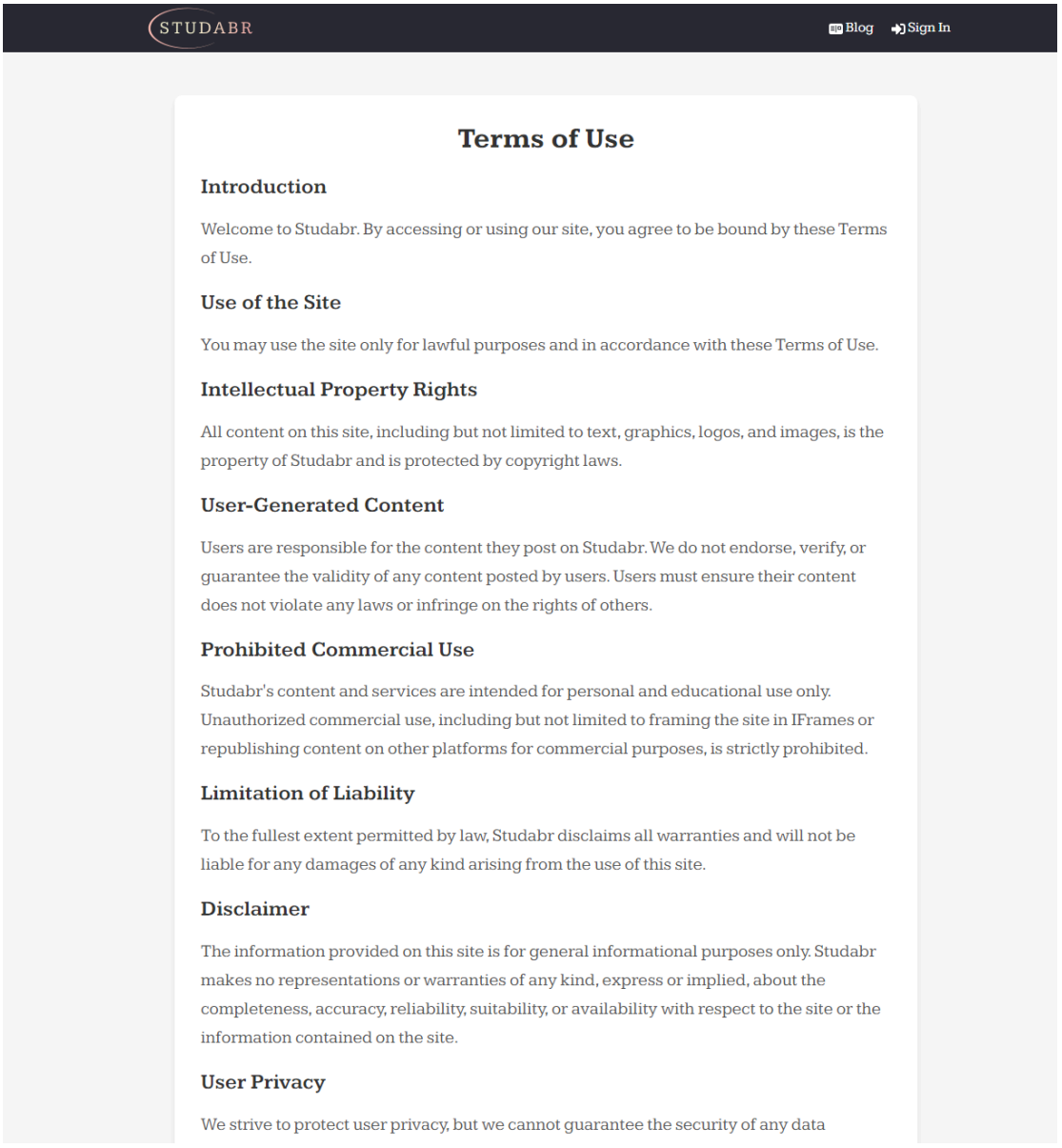


Рис. 11. Перша частина сторінки “Умови Використання”

- Друга частина сторінки:

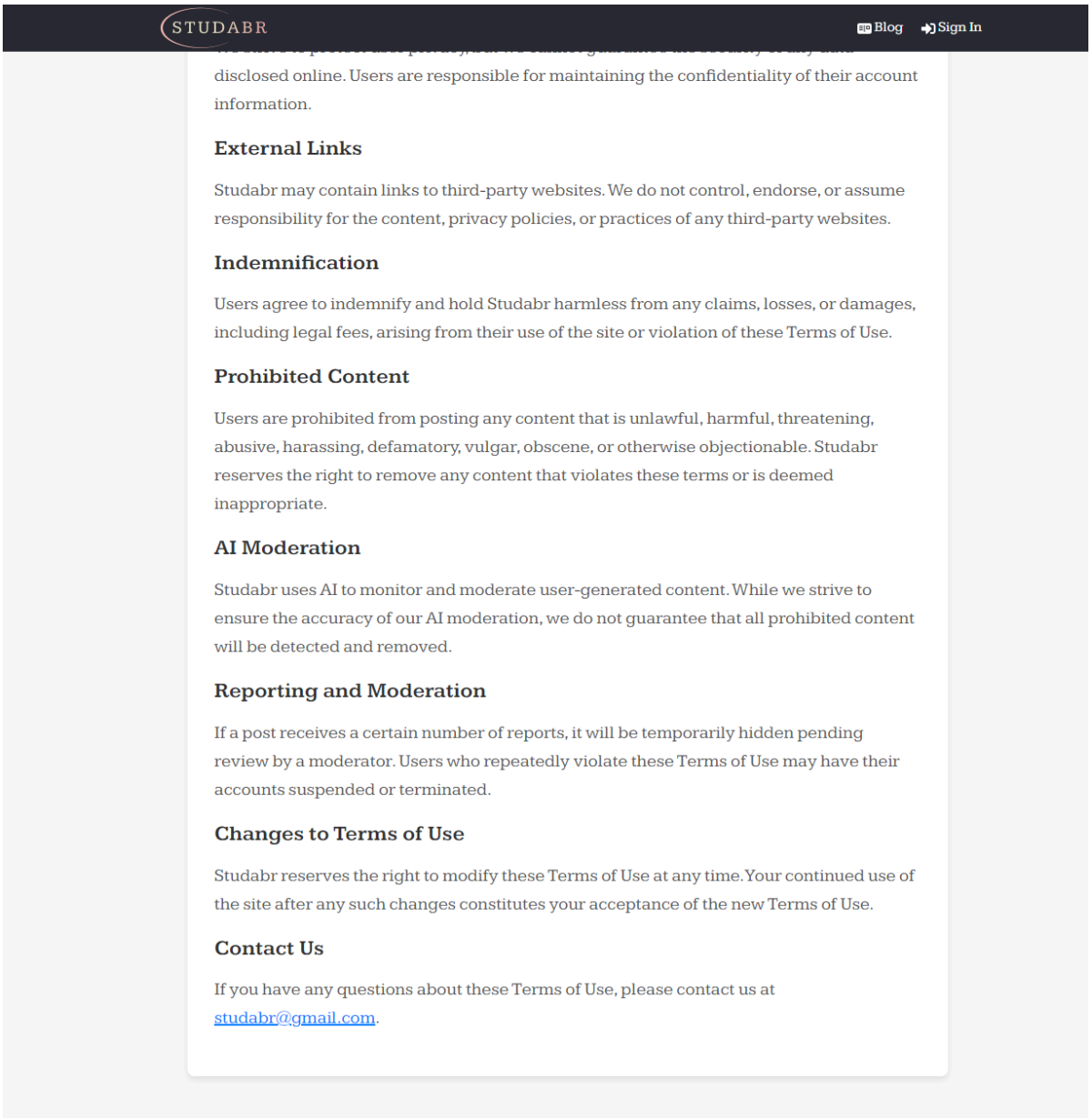


Рис. 12. Друга частина сторінки “Умови Використання”

- Політика конфіденційності (Privacy Policy) - Ця сторінка описує які дані користувача збирає застосунок, як він їх оброблює та використовує. Також сторінка описує захист особистих даних користувача та його права. В цьому випадку теж було проведено дослідження юридичної сторони цього питання, для того, щоб зняти з мене, як з розробника зайву відповідальність

- Перша частина сторінки:

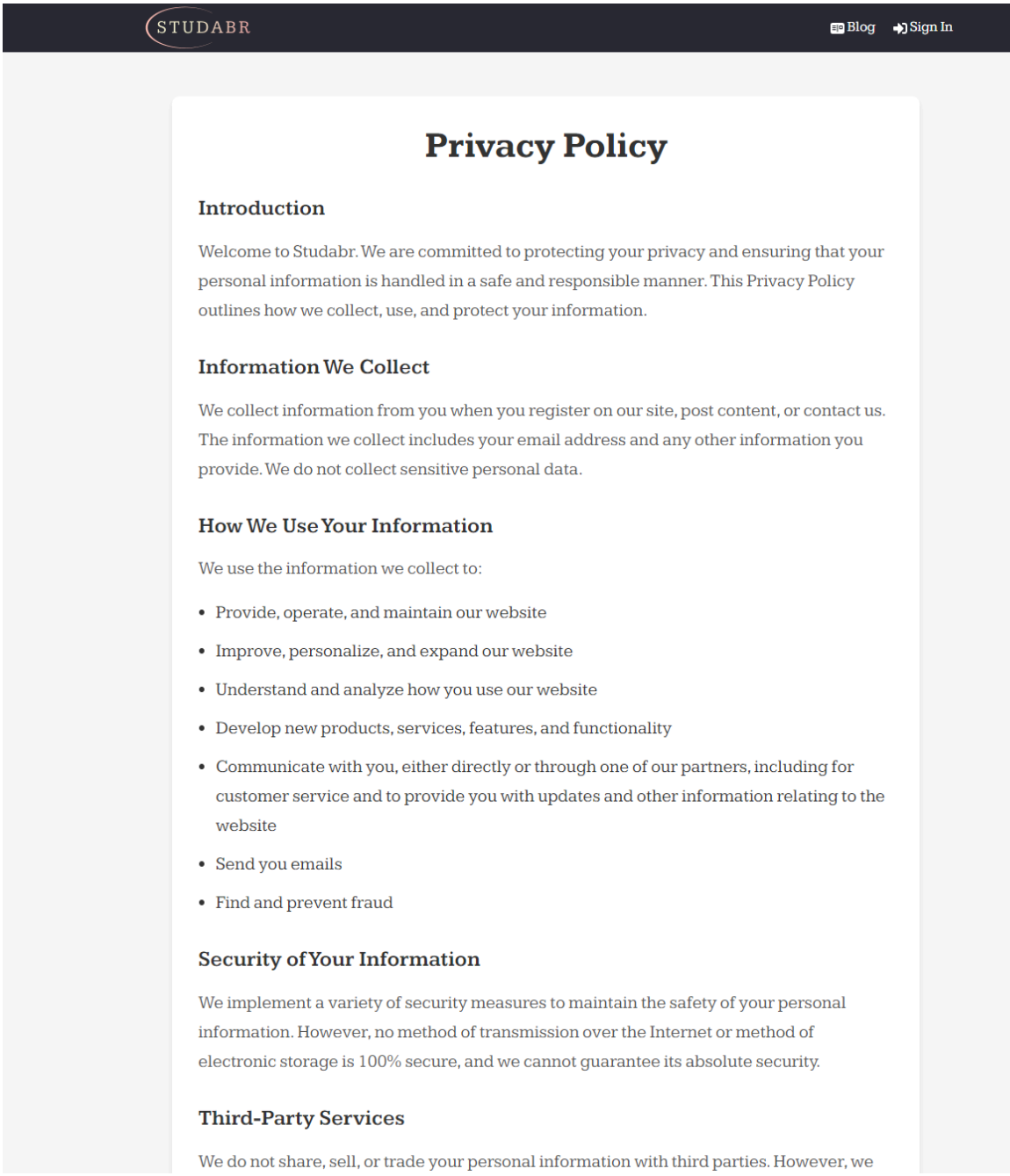


Рис. 13. Перша частина сторінки “Політика Конфіденційності”

- Друга частина сторінки:

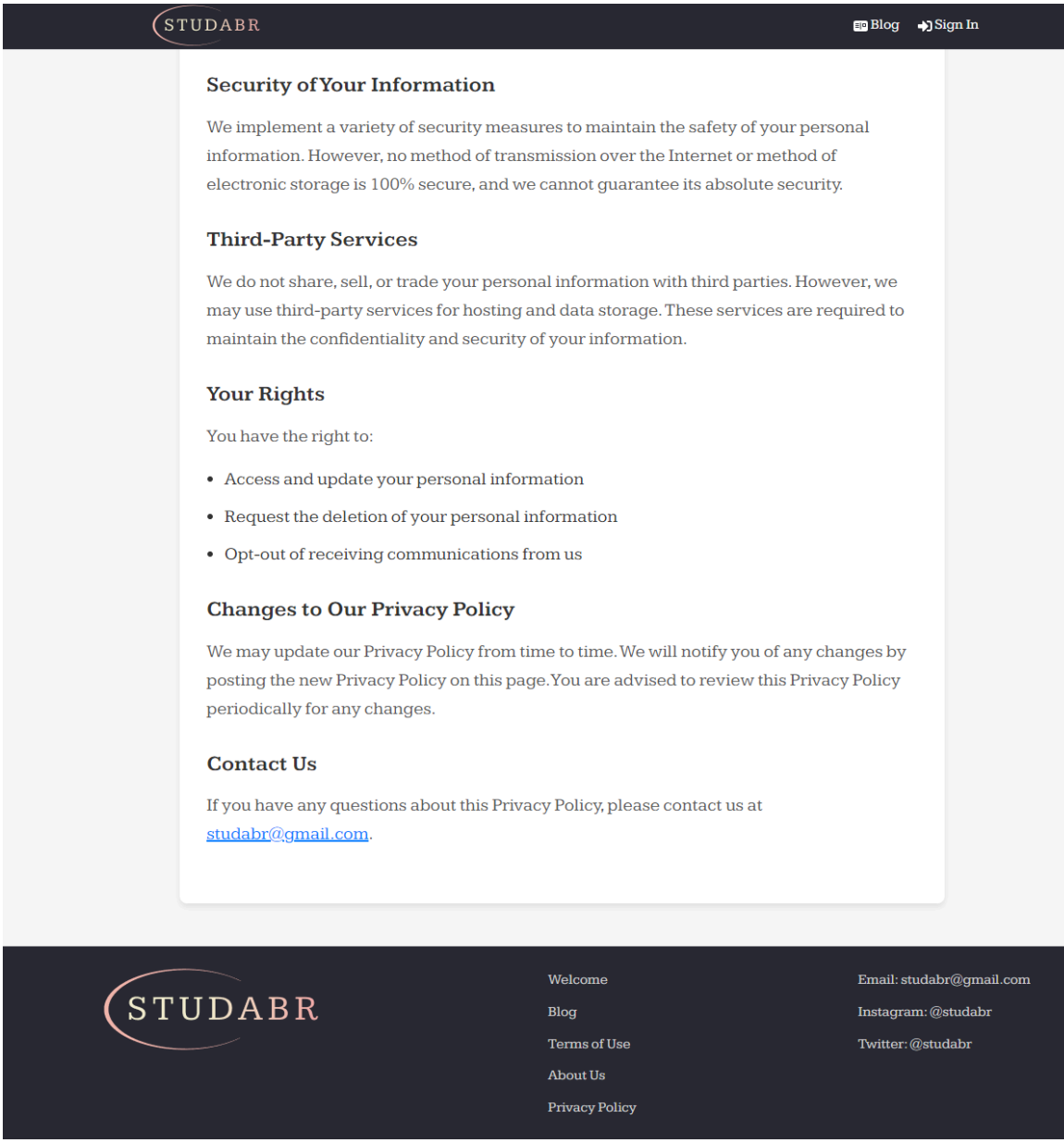


Рис. 14. Друга частина сторінки “Політика Конфіденційності”

- Сторінка авторизації (Sign In) - це сторінка для авторизації користувача. На ній користувач може ввести свої поштову адресу та пароль, або перейти на сторінку реєстрації, у разі якщо раніше він не реєструвався.
  Функціональність:
  - Валідація: ми валідуємо поле для поштової адреси за допомогою регулярного виразу:
  - `^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$`

- Використання сервісу користувачів: При натисканні кнопки “Sign In” ми викликаємо метод login для відправлення даних на перевірку до серверу, а також отримання JWT-token, після чого, у разі успіху ми створюємо cookie з цим токеном та переадресовуємо користувача на сторінку зі списком публікацій. У разі невдалої спроби, ми повідомляємо користувача про некоректність введених даних, та просимо спробувати знову.

Сторінка:

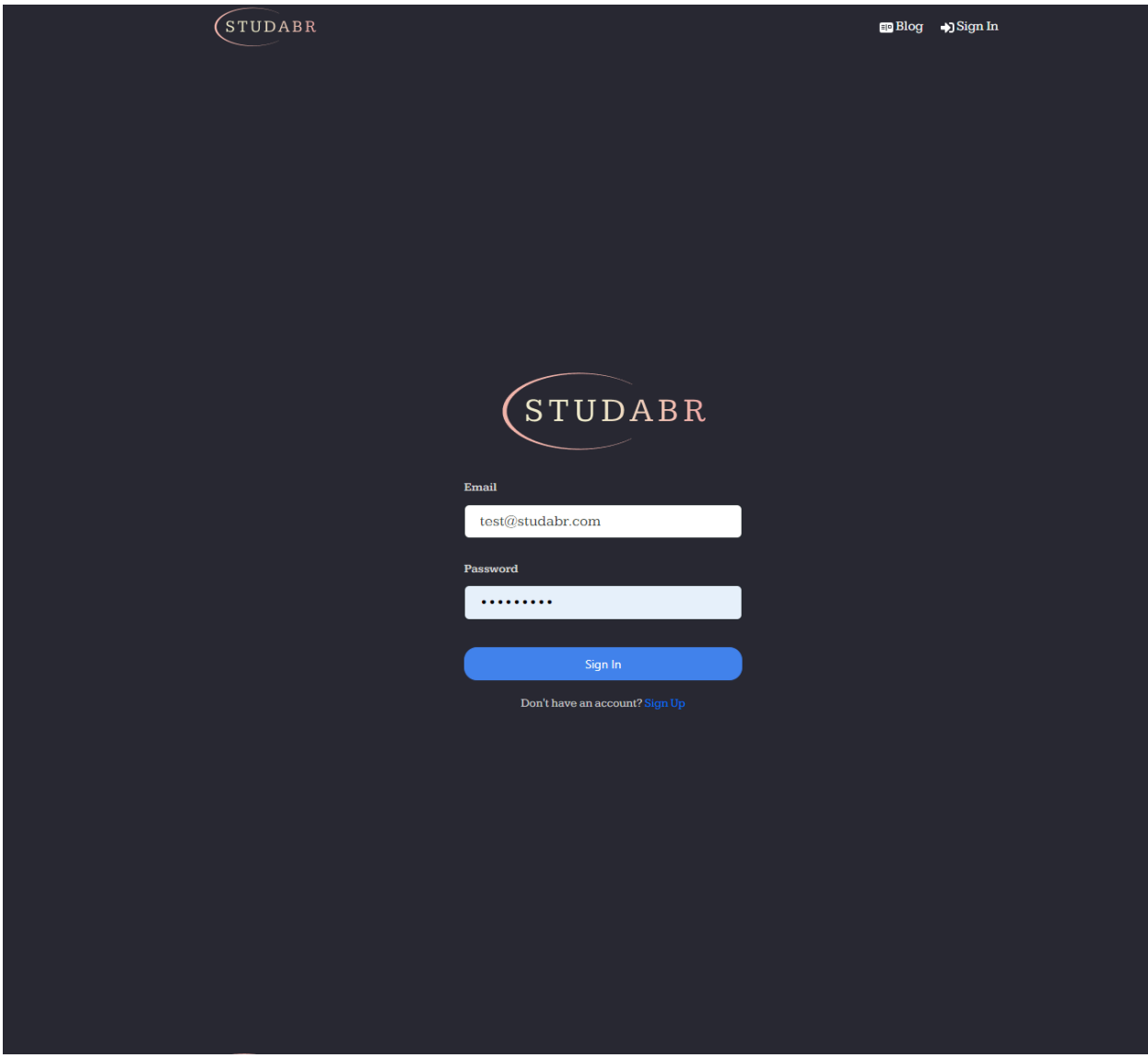


Рис. 15. Сторінка “Авторизації”

- Сторінка реєстрації (Sign Up) - ця сторінка розроблена для реєстрації нових користувачів. На ній можна заповнити форму своїми даними та створити акаунт на Студабр. У разі якщо у користувача вже є акаунт то він може перейти за посиланням на

сторінку авторизації.
Функціонал:

- Валідація: Поля електронної пошти та паролю валідуються за допомогою регулярних виразів. Пароль має містити в собі хоча б одну велику літеру, одну цифру та мінімум 8 символів. Регулярні вирази:

```
const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

const passwordRegex = /^(?=.*[A-Z])(?=.*\d).{8,}$/;
```

- Використання сервісу користувачів: Після натискання кнопки “Sign Up” та успішної валідації форми, ми викликаємо метод register для відправлення даних на сервер, де буде створено нового користувача, а також для отримання JWT-token, який буде збережено в cookie.

  Сторінка:

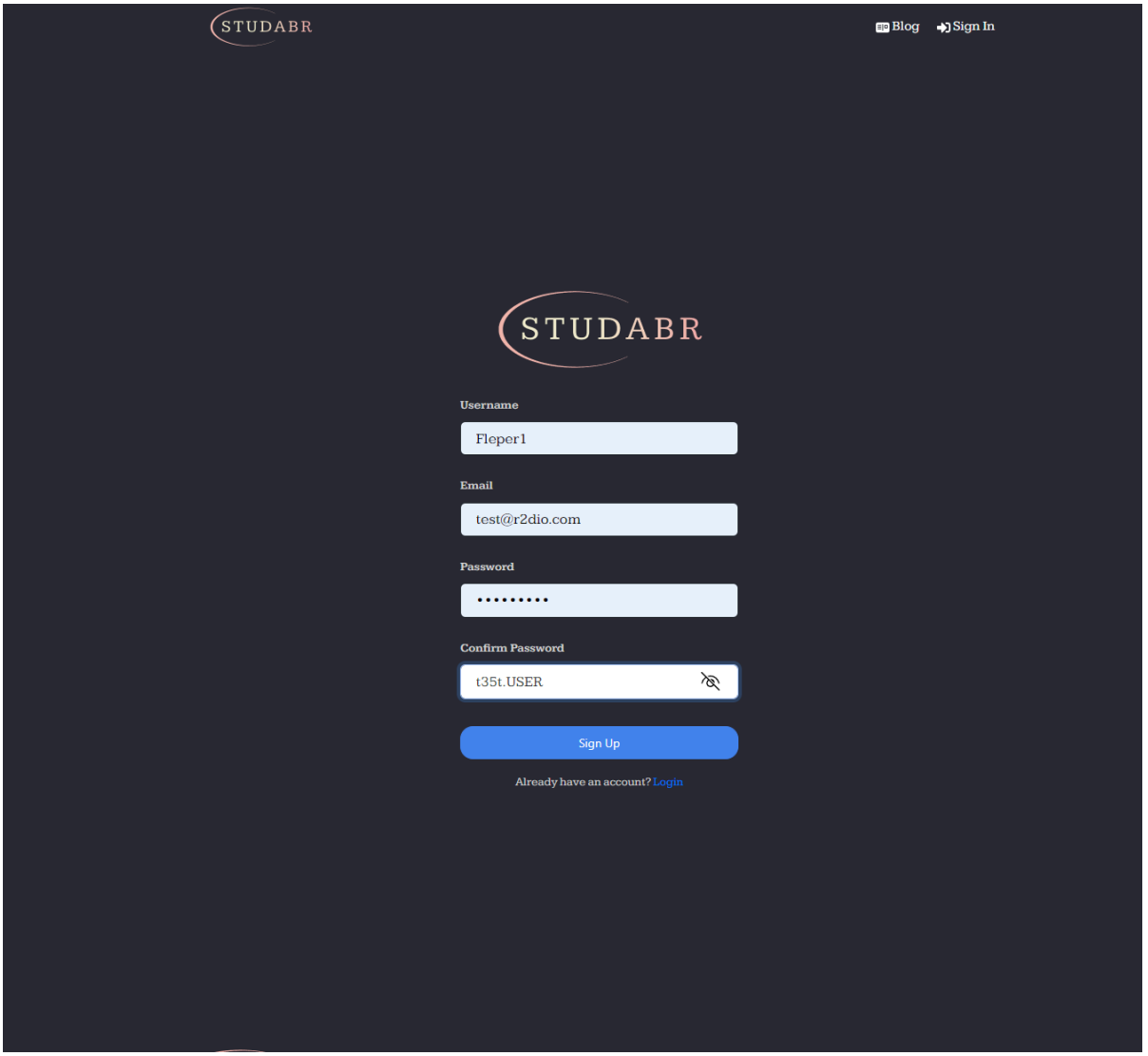


Рис. 16. Сторінка “Реєстрації”

- Сторінка публікацій(Blog) - ця сторінка є серцем сайту Студабр, бо саме вона грає головну роль в пошуку корисної інформації та навчанні. На цій сторінці користувач може ознайомитись зі списком публікацій, а також налаштовувати фільтри для пошуку потрібної інформації. Також на сторінці є пагінація, що дозволяє зменшити кількість часу потрібного для відображення сторінки. Сторінка:

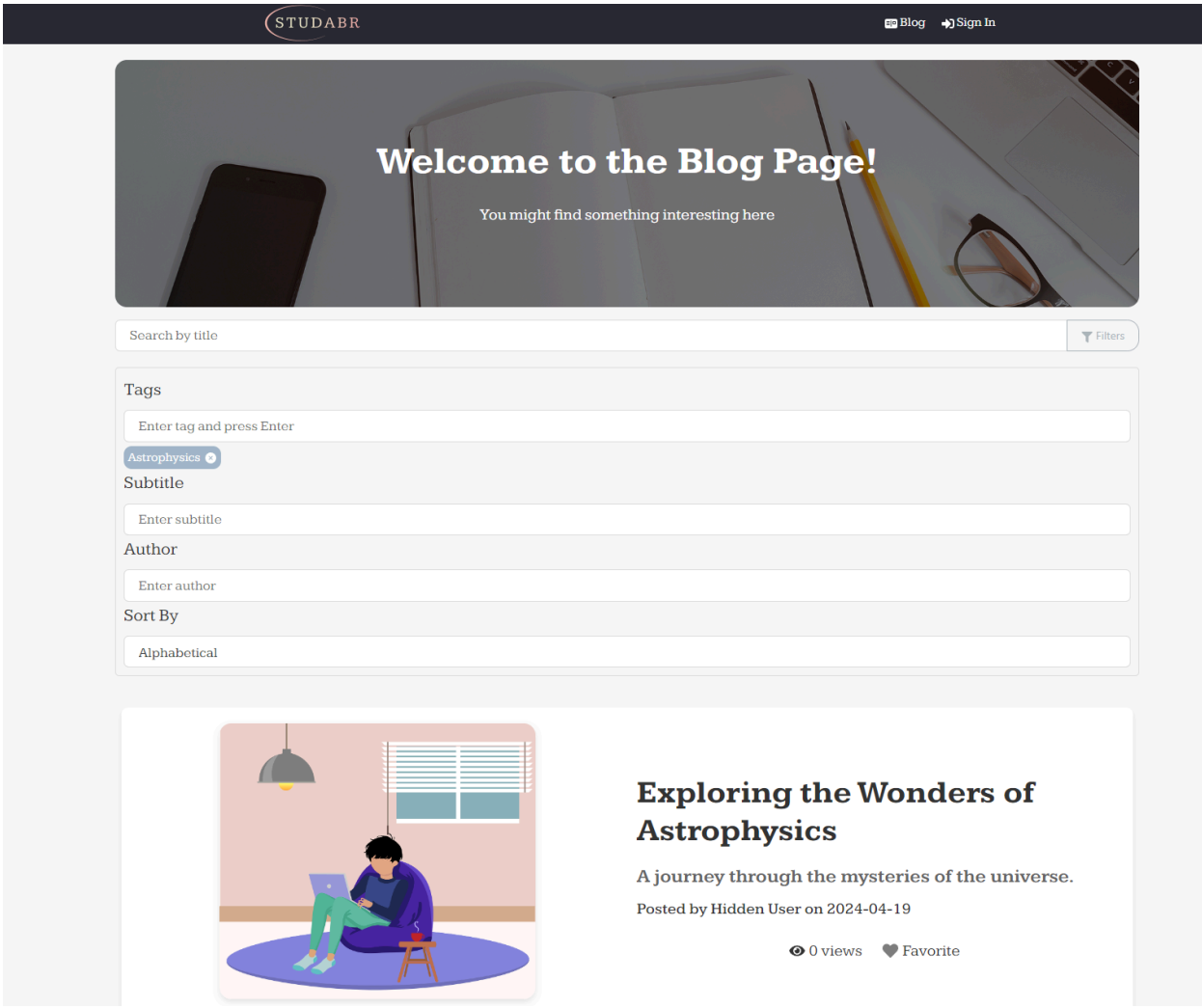


Рис. 17. Сторінка “Публікацій”

Сторінка має в собі багато функціонала:

  - Використання сервісу публікацій: коли користувач заходить на сторінку, ми виконуємо метод fetchPublications, щоб отримати усі наявні на сервері публікації, після чого вони додатково обробляються на клієнті, для створення пагінації.
  - Використання пошукових запитів: Ми маємо компонент для пошуку за заголовком

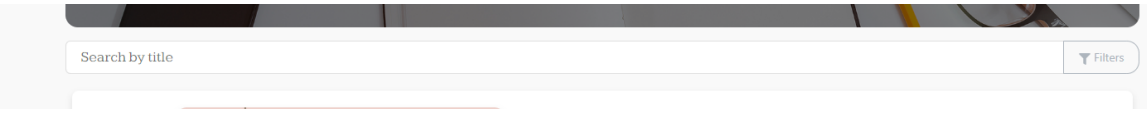


Рис. 18. Компонент “Пошуковий рядок”

Щоб покращити швидкість та зменшити навантаження, було застосовано алгоритм з тайм-аутами, щоб дочекатися поки користувач припинить вводити пошуковий запит і тільки після цього виконувати його. Тобто поле буде очікувати протягом 1 секунди дій від користувача, у разі якщо цих дій не відбулось, то обробить наявний запит. Також тут розташована кнопка “Filters”, яка відкриває додаткові варіанти фільтрації.

- Додаткові фільтри: В більшості випадків користувачу при пошуку чогось конкретного потрібно більше можливостей для налаштування. Для цього був створений компонент з додатковими фільтрами. Він не відображається на екрані аж поки користувач не натиснув кнопку “Filters”. За їх допомогою користувач може визначити більш конкретно свої запити

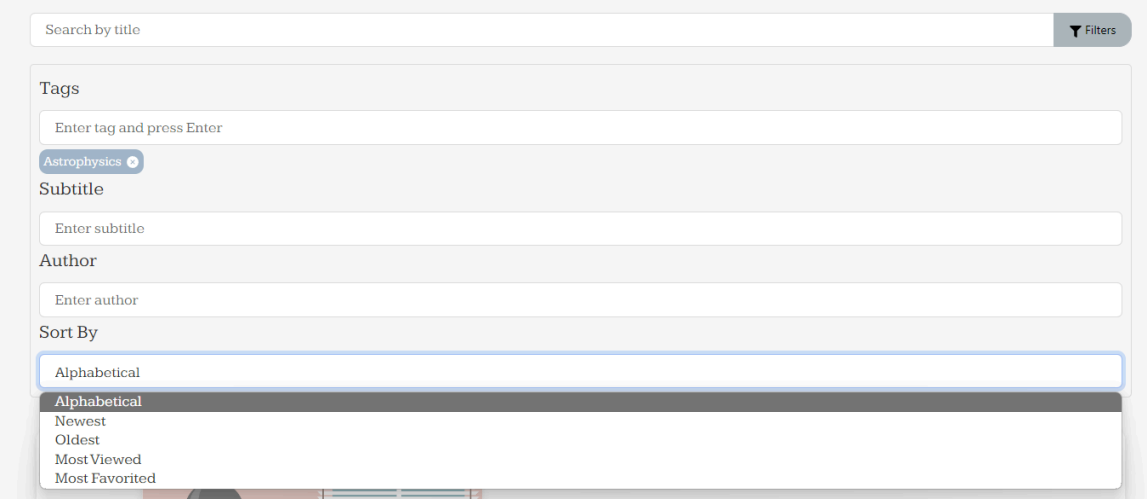


Рис. 19. Компонент “Додаткові Фільтри”

Тут реалізована функціональність для пошуку за тегами, підзаголовками, ім’ям автору, а також додана можливість сортування. Вся ця функціональність також дочікується поки користувач закінчить свій запит і лише після цього починає обробляти вказані фільтри.

- Перед перегляд публікації: основна задача цього компоненту надати користувачу стислий формат інформації про публікацію для кращого розуміння того про що йде мова в самій публікації, таким чином ми робимо

процес пошуку потрібної інформації більш зручним та швидким для користувачів. Також тут відображається кількість переглядів та кнопка вподобайки, щоб зробити інтерфейс ще більш зручним.

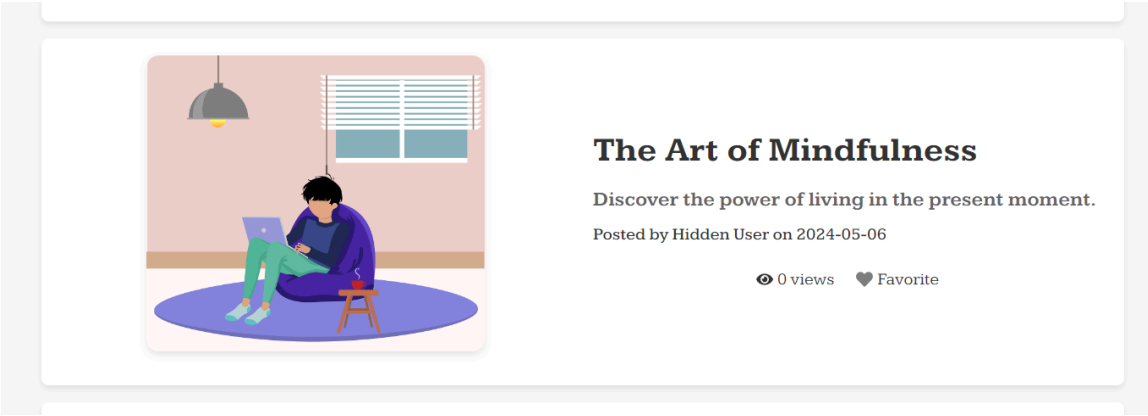


Рис. 20. Компонент “Перед перегляд публікації”

Також тут додано логіку для перенаправлення користувачів на окремі сторінки публікацій, у разі якщо вони клікнули на одну із них. Для більшого розуміння що на неї можна натиснути курсор перетворюється в click, а також відбувається анімація при наведенні.

- Пагінація: щоб оптимізувати роботу сторінки, була додана пагінація. Якщо відображати одразу всі публікації за раз, то час на відображення сторінки буде занадто великий, через що користувач може просто покинути застосунок або відволіктися на щось інше.

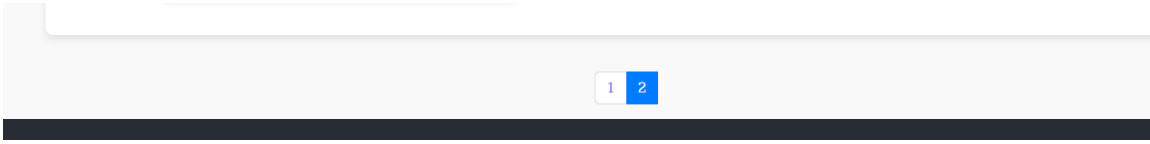


Рис. 21. Компонент “Пагінація”

Була додана функціональність яка рахує скільки потрібно сторінок для відображення всіх публікацій, та динамічно створює кнопки з номерами сторінок, після кліку на номер сторінки вона оновлює масив постів для відображення.

- Сторінка публікації(Post): саме ця сторінка відповідає за надання обмін корисною інформацією між користувачами. Вона має невелику функціональність, але залишається однією з найважливіших сторінок на сайті Студабр.

Сторінка:

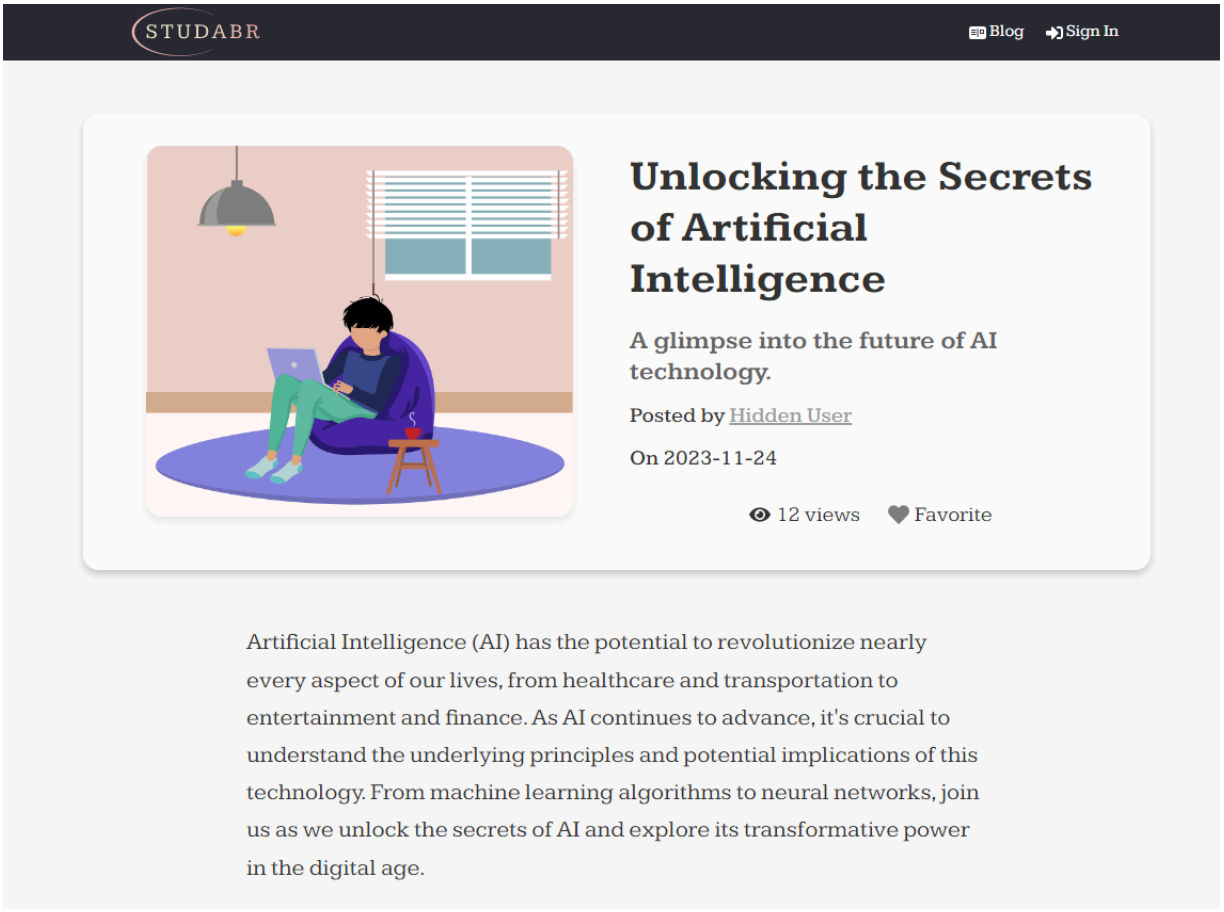


Рис. 22. Сторінка “Публікація”

Функціональність:

- Обробка URL: на цій сторінці додана логіка, яка отримує slug з URL, для подальшого використання в отриманні даних про публікацію.
- Використання сервісу публікацій: ми викликаємо метод fetchPublication для отримання публікації.
- Отримання контенту публікації: після отримання публікації треба підвантажити її markdown контент та відобразити його. Тому ми беремо назву файлу, в якому зберігається контент публікації, робимо з нього посилання на сховище файлів Google Cloud Store, та отримуємо markdown контент. Після цього відображаємо його за допомогою бібліотеки React Markdown.
- Перегляди: кожен раз коли користувач потрапляє на сторінку публікації ми відправляємо запит на сервер, щоб оновити кількість переглядів у публікації.
- Вподобайки: на сторінці є кнопка Favorite, яка використовує відповідний сервіс для відправлення запиту

на сервер, щоб там оновити список публікацій які сподобалися користувачеві.

- Навігаційний заголовок сайту (Header): це компонент який ми користувач бачить на кожній сторінці сайту. Його головною метою є надання можливості швидко перемикатись між основними сторінками застосунку. В ньому додана логіка яка закріплює його зверху і тому користувач завжди зможе їх скористатися.

Рис. 23. Компонент “Навігаційний заголовок”

- Навігаційний підвал застосунку (Footer): цей компонент дуже схожий на Header. Він завжди розташований у кінці сторінки. У Footer перелічені усі сторінки сайту Студабр з посиланнями на них, а також соціальні мережі Студабр.

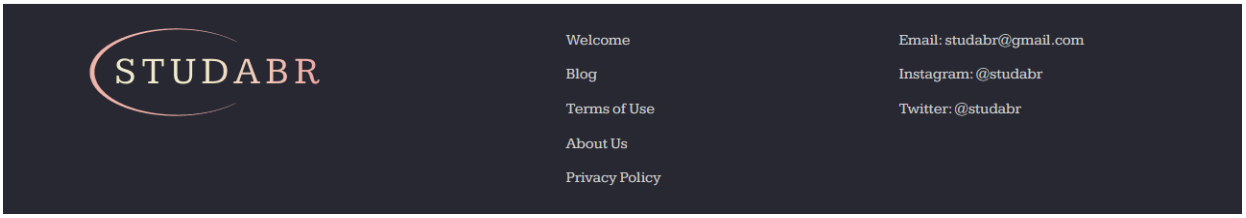


Рис. 24. Компонент “Навігаційний підвал застосунку”

- Текстовий редактор: це основний інструмент сайту Студабр. Він впроваджує логіку для написання контенту публікацій. Має багато функціональності та декілька способів використання у виді компонента з налаштуваннями.
  Редактор:

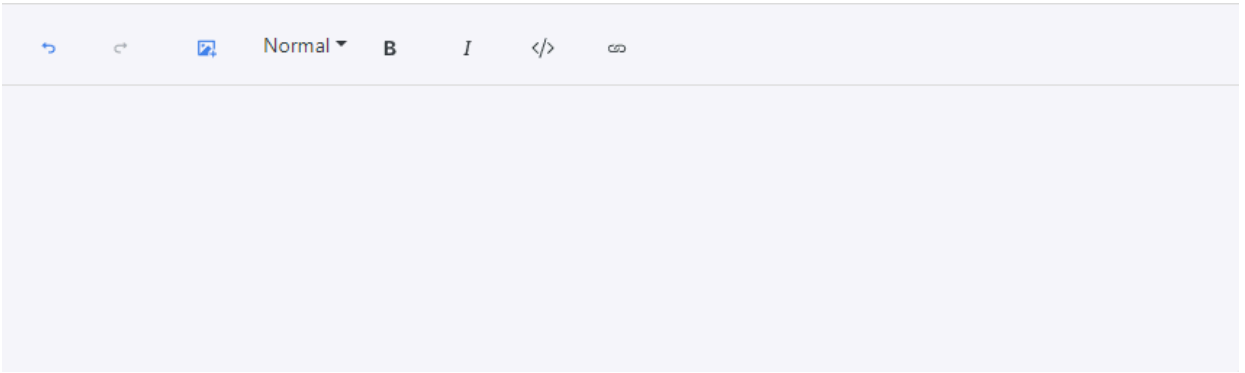


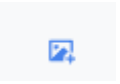
Рис. 25. Компонент “Редактор тесту”

- Функціональність:
  - Підтримка картинок: було розроблено окремий модуль, який дозволяє зберігати та відображати картинки у редакторі.
    - Бібліотека Lexical працює з редактором через “вузли”, таким чином вона надає нам набір функцій та типів для створення та налаштування власних вузлів. Використовуючи ці функції, я створив вузол для картинок, який вміє їх відображати.

```
export const IMAGE: TextMatchTransformer = {
  dependencies: [ImageNode],
  export       : node => {
    if (!$isImageNode(node)) {
      return null;
    }

    return `![${node.getAltText()}](${node.getSrc()})`;
  },
  importRegExp: /!(?:\[([^[]*)\])(?:\(([^(]+)\))/,
  regExp      : /!(?:\[([^[]*)\])(?:\(([^(]+)\))$/,
  replace     : (textNode, match) => {
    const [, altText, src] = match;
    const imageNode = $createImageNode({
      altText,
      maxWidth: 800,
      src,
    });
    textNode.replace(imageNode);
```

```
  },
  trigger: ")",
  type    : "text-match",
};
```

- Коли користувач завантажує картинку  , то вона переводиться в base64 рядок та створюється вузол картинки з цією строкою у форматі markdown, base64 використовується у якості посилання на картинку. Також картинки хешуються для подальшого використання.
- Після обробки та створення вузла, він передається до бібліотеки Lexical. Таким чином текстовий редактор використовує вказаний компонент для відображення картинки.
- Коли контент зберігається я звертаюся до захешованих картинок, щоб отримати їх base64 код, переводжу у масив байтів, на його основі генерую назву картинки. Після цього відправляю ці дані на сервер для зберігання у сховищі, у разі помилки, повідомляю користувача про те, що щось пішло не так.
  У разі коли збереження пройшло успішно, за допомогою регулярного виразу я в маркдауні заміняю base64 картинки на посилання на ці картинки в сховищі:

```
export async function uploadBase64Images(markdown:
string, imagesCache: Array<string>) {
  const uploadPromises = imagesCache.map(async src => {
    if (!src.startsWith("data:image/")) {
      return;
    }
    const typeStart = src.indexOf("image/") +
"image/".length;
```

```
    const typeEnd = src.indexOf(";base64,");

    const type = `.${src.slice(typeStart, typeEnd)}`;



    const res = await uploadAsset({ fileContent:
src.split(",")[1], type });

    markdown = markdown.replace(src, res.fileUrl);

  });



  await Promise.all(uploadPromises);



  return markdown;

}
```

- Підтримка автоматичного зберігання даних: для випадку коли користувач редагує вже створений пост, було створено віджет, який автоматично оновлює дані, у випадку коли користувач нічого не змінює впродовж 2 секунд. Це прискорює процес редагування, та робить його більш приємним та зручним.
  Ось приклад цього віджету у двох випадках

- Коли дані оновлено:

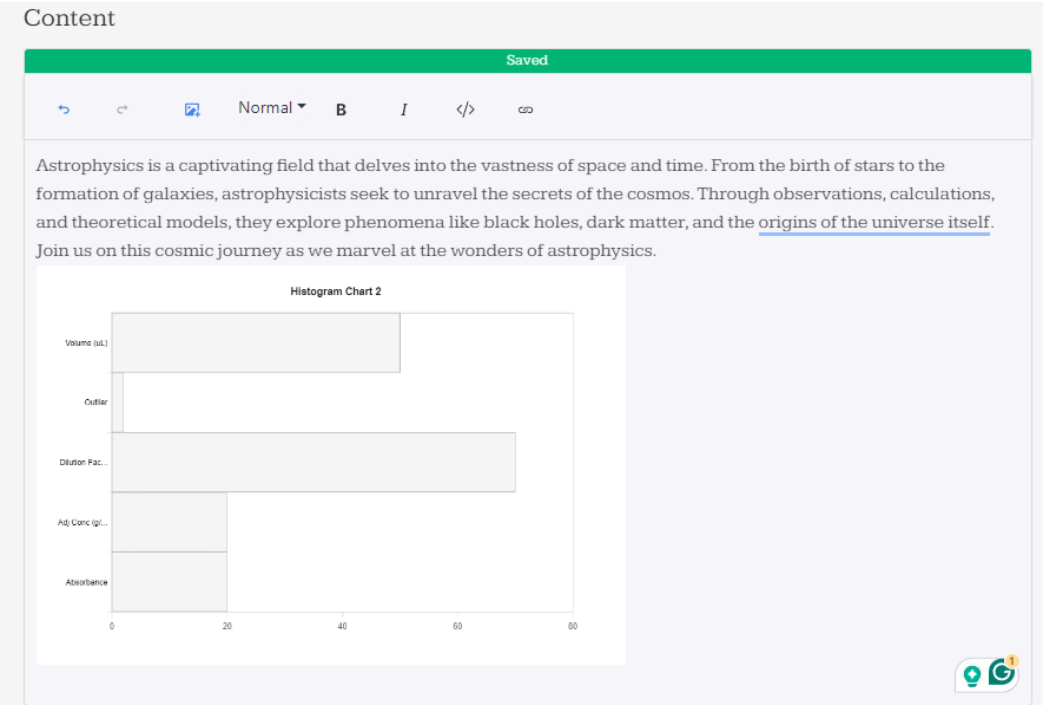


Рис. 26. Компонент “Редактор тексту з оновленням даних”

- Коли дані в процесі оновлення:

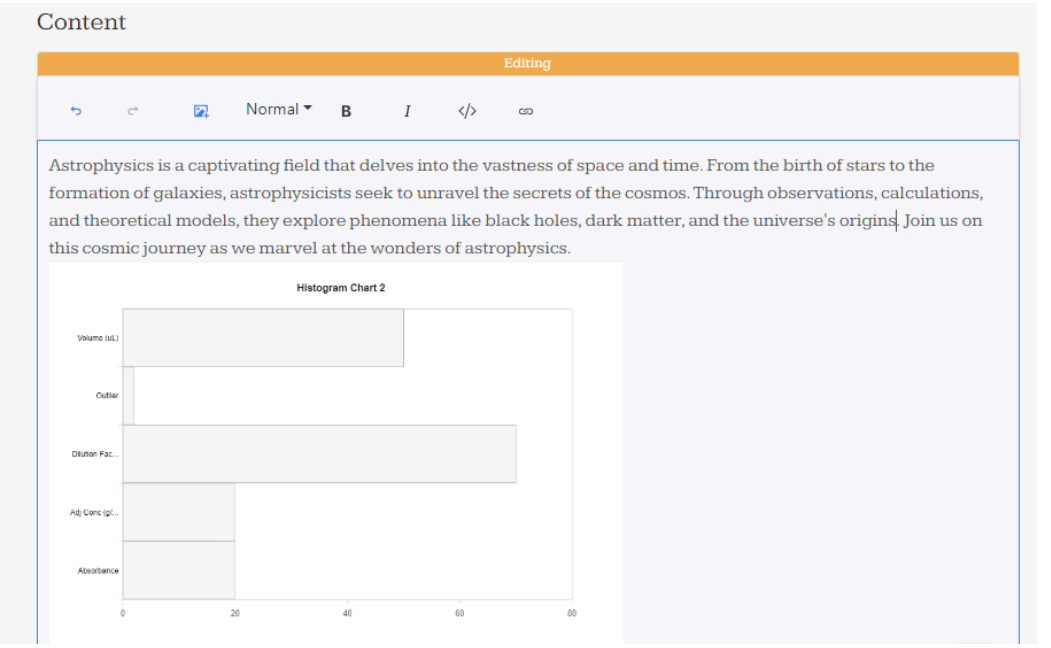
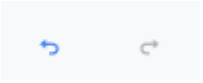


Рис. 27. Компонент “Редактор тексту під час оновлення”

- Підтримка кнопок Скасувати/Відтворити: натискання цих кнопок дозволяє нам повернутися до попереднього кроку, або на крок вперед, якщо це можливо. Для реалізації було використано історію станів, яку надає бібліотека Lexical. За її допомогою я додав можливість змінювати поточний стан на крок вперед або назад  . Цей функціонал робить редактор більш зручним та сучасним.

- Підтримка форматування: були додані базові методи форматування тексту, які підтримуються markdown, для забезпечення гнучкості та більшого контролю при створенні контенту публікації. Форматування також використовує логіку вузлів, а також відстежує виділений текст:
  - Заголовки та списки: у користувача є можливість використовувати різні заголовки, а також два базові види списків - нумерований та маркований:

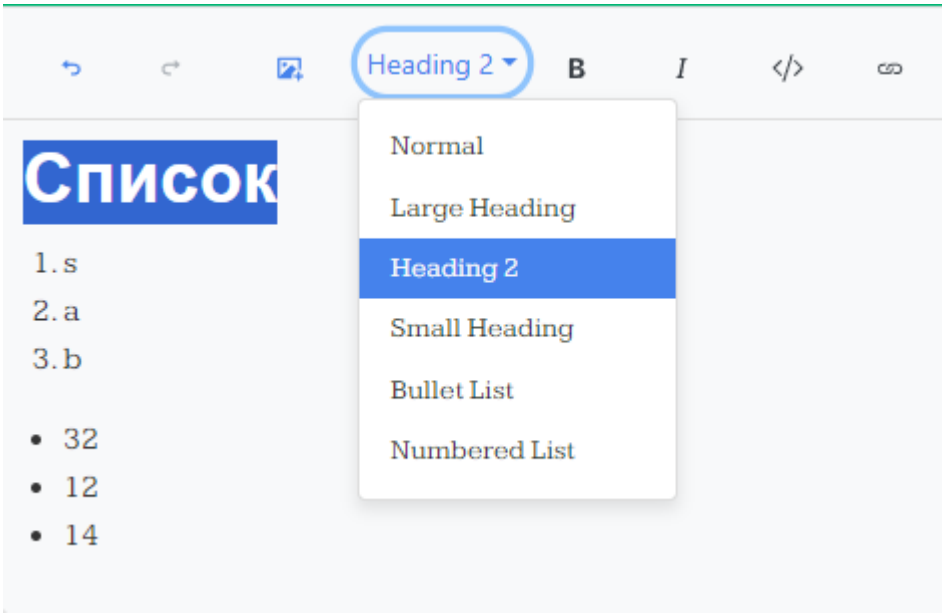


Рис. 28. Ілюстрація “Форматування заголовків та списків”

- Жирний текст: користувач може зробити будь-яку частину тексту жирною:

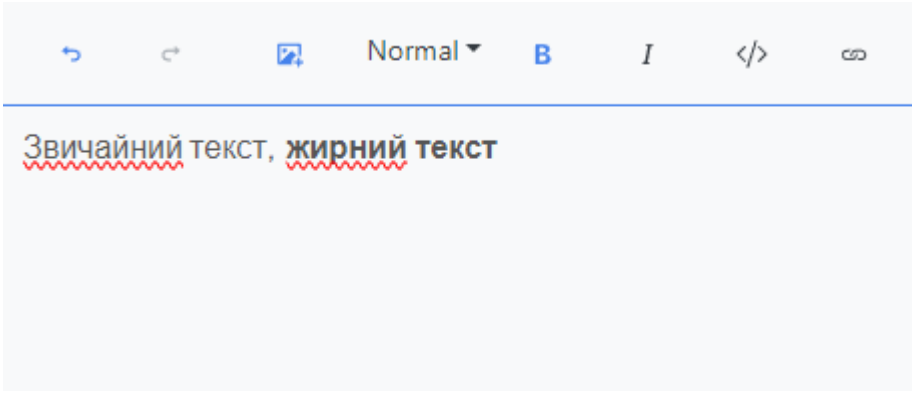


Рис. 29. Ілюстрація “Налаштування жирного тексту”

- Курсивний текст: користувач може написати будь-яку частину тексту курсивом:

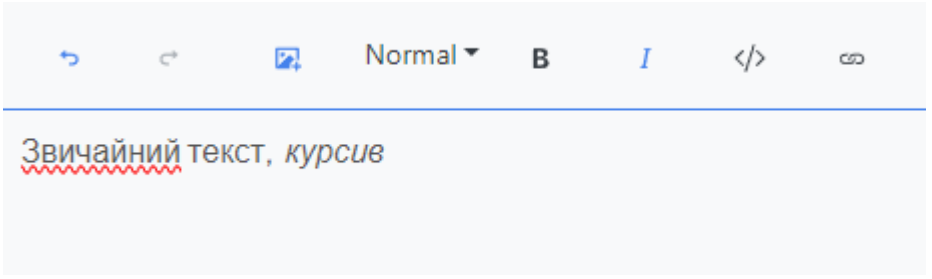


Рис. 30. Ілюстрація “Налаштування курсиву”

- Виділений текст: користувач може змінити колір частини тексту, для його виділення серед іншого:

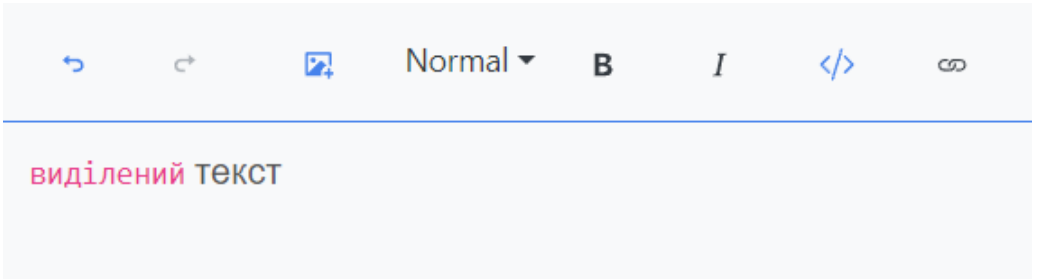


Рис. 31. Ілюстрація “Налаштування виділення тексту”

- Посилання: користувач може вставляти посилання в текст, щоб мати можливість вказати джерело

інформації тощо:

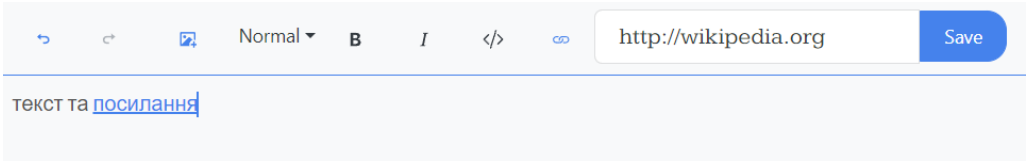


Рис. 32. Ілюстрація “Налаштування посилання у тексті”

- Створення\Редагування публікації (Create/Edit Post): на цій сторінці користувач може створювати нові або редагувати наявні публікації за допомогою форми з основними полями публікації, а також текстового редактора. Ця сторінка використовує однаковий компонент, але реалізована логіка, яка визначає за допомогою URL який віджет текстового редактору треба використати, а також, чи потрібно робити запит на сервер для отримання наявних даних про пост.
  Розберемо більш детально функціональність в обох випадках:
  - Створення: сторінка використовує віджет без автоматичного зберігання контенту публікації, щоб запобігти непотрібним запитам на сервер та зберігання даних. Після натискання на кнопку “Create Post” дані з форми збираються та відправляються до методу createPublication у сервісі публікацій. У разі успішного створення публікації на сервері, користувача перекидає на сторінку зі списком публікацій. У разі помилки користувач отримує повідомлення про те, що щось пішло не так.
  - Редагування: сторінка отримує slug публікації та звертається до методу fetchPublication з сервісу публікацій. У разі помилки, користувач отримує повідомлення про це. У разі успіху форма заповнюється наявною інформацією. Щодо контенту, то назва файлу, отримана з сервера, перетворюється на посилання на цей файл у сховищі, після цього виконується запит на отримання даних з контенту файлу. Отриманий контент передається у текстовий редактор у якості значення для ініціалізації. Для текстового редактора використовується віджет з автоматичним оновленням даних на сервері під час редагування, тож користувачу не потрібно натиснути кнопку зберегти, кожного разу, коли він вніс невеликі зміни.

Варто зазначити, що користувач має перемикач віджета, на випадок, коли він не хоче, щоб дані самостійно оновлювались.

- Вікно користувача(User Poppup): у цьому вікні користувач може побачити базову інформацію про себе, а також скористатися кнопками “Change Password” та “Sign Out”.

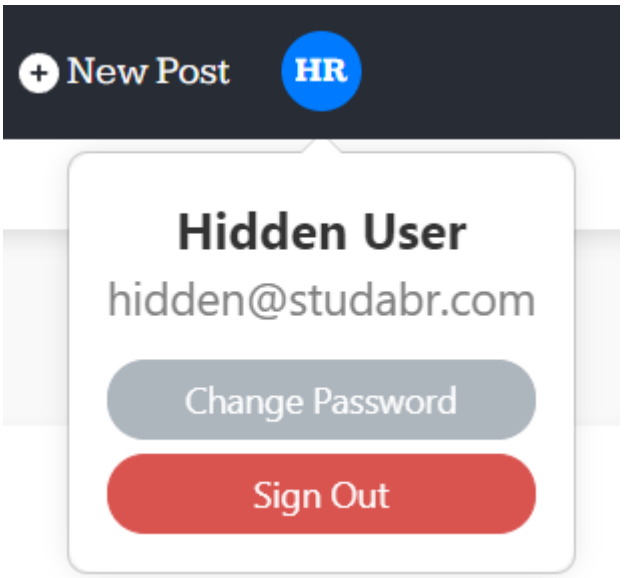


Рис. 33. Компонент “Вікно користувача”

  - Змінити пароль: у разі натискання на цю кнопку, користувача буде направлено на сторінку підтвердження, де він має ввести спочатку свій поточний пароль, після цього його буде направлено до форми зміни пароля. У кінці, коли користувач натискає “Submit” дані надсилаються до сервера, а користувача перекидає на “Вітальну” сторінку.

4. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ КОРИСТУВАЦЬКОЇ ЧАСТИНИ САЙТУ СТУДАБР

4.1. Методи та інструменти тестування

1. Методи тестування:

    - Юніт-тестування: Тестування окремих модулів або функцій у відриві від інших частин системи. Юніт-тести перевіряють правильність роботи індивідуальних блоків коду, забезпечуючи надійність та стабільність додатка на низькому рівні.
    - Інтеграційне тестування: Тестування взаємодії між різними модулями та компонентами. Інтеграційні тести перевіряють, чи правильно взаємодіють між собою різні частини системи, виявляючи проблеми з інтеграцією та комунікацією між модулями.

2. Інструменти тестування:

    - Mocha: Основний інструмент для юніт та інтеграційного тестування. Mocha забезпечує гнучкий та розширюваний фреймворк для тестування з підтримкою асинхронних тестів.
    - Chai: Використовується разом з Mocha для написання тверджень (assertions). Chai забезпечує різні стилі тверджень, включаючи BDD та TDD, що дозволяє писати читабельні та зрозумілі тести.
    - React-test-renderer: Інструмент для юніт-тестування React-компонентів. React-test-renderer дозволяє рендерити компоненти в пам'яті та перевіряти їх стан без взаємодії з DOM.
    - @testing-library/react: Бібліотека для тестування React-компонентів, яка зосереджується на тестуванні поведінки додатка з точки зору користувача. @testing-library/react забезпечує легке та інтуїтивне API для взаємодії з компонентами через їх рендеринг, знаходження елементів та обробляє різні дії, наприклад натискання кнопок та інше. Бібліотека дозволяє написання тестів, які не залежать від внутрішньої реалізації компонентів, що робить їх більш надійними та зручними для підтримки.

4.2. Результати тестування

1. Юніт-тести:

   Весь основний функціонал (генерація динамічної URL на сховище даних, переведення картинки до масиву байтів, сервіси для спілкування з сервером та інше) успішно пройшли юніт-тести.

   Приклад:

```
import { expect }                from "chai";

import { getStaticStorageLink } from "../get-static-storage-link";



describe("getStaticStorageLink", () => {

  it("should throw an error", async () => {

    const rejectMessage = "Storage link(process.env.DEV_STORAGE_LINK) is
not defined, please check your .env file";

    try {

      getStaticStorageLink();

    } catch (e: any) {

      expect(e.message).to.be.equals(rejectMessage);

    }

  });

  it("should return dev storage link", async () => {

    process.env.DEV_STORAGE_LINK =
"http://127.0.0.1:10000/devstoreaccount1/blog-engine";

    const link = getStaticStorageLink();
```

```
expect(link).to.be.equals("http://127.0.0.1:10000/devstoreaccount1/blog-engine");
  });
});
```

2. Інтеграційні тести:

   Проведено інтеграційне тестування для перевірки роботи сторінок та їх взаємодії.

   Приклад:

```
import React from 'react';
import { render } from '@testing-library/react';
import { expect } from 'chai';
import App from './App';
import { Router } from 'wouter';
import { MemoryRouter } from 'react-router-dom';
import welcomeText from './welcomeText.ts'
import blogText from './blogText.ts'


describe('App Component', () => {
  it('renders WelcomePage component by default', () => {
    const { getByText } = render(
      <MemoryRouter>
        <App />
      </MemoryRouter>
    );
```

```
    expect(getByText(welcomeText)).to.exist;
  });


  it('renders BlogPage component when path is /blog', () => {
    const { getByText } = render(
      <MemoryRouter initialEntries={['/blog']}>
        <App />
      </MemoryRouter>
    );


    expect(getByText(blogText)).to.exist;
  });
});
```

3. Результати тестування:

   Тестування застосунку, його основного функціонала, компонентів та сторінок пройшло успішно!

   - Виявлення та виправлення помилок: У ході тестування були виявлені кілька помилок та неочікувана поведінка в деяких функціях та компонентах. Це дозволило мені вчасно внести необхідні виправлення в коді та забезпечити стабільну роботу системи.
   - Покращення продуктивності: У ході тестування були визначені потенційні вузькі місця в продуктивності деяких компонентів. Після оптимізації алгоритмів, розбиття компонентів на окремі частини і покращення ефективності коду, час виконання

основних операцій значно зменшився, що призвело до підвищення загальної продуктивності сайту Студабр.
- Підвищення стабільності: Завдяки інтеграційному тестуванню було перевірено коректну взаємодію між різними сторінками та компонентами. Це дозволило виявити і виправити проблеми з маршрутизацією і передачею даних між компонентами, забезпечивши плавний користувацький досвід.

Загалом, весь функціонал проекту працює коректно й так, як це і очікувалось. Виконане тестування забезпечило надійність і стабільність роботи сайту, що гарантує позитивний досвід користувача.

4.3. Впровадження та розгортання користувацької частини

1. Підготовка до розгортання:
    - Очищення коду: Видалення непотрібного коду, коментарів та тимчасових файлів.
    - Форматування коду: виклик npx prettier –write, для видалення непотрібних відступів, невикористовуваних змінних та іншого.
    - Оновлення документації: внесення опису усього важливого функціоналу проекту та кроків які треба виконати перед його запуском, це робиться для більшого розуміння іншими розробниками.
    - Підготовка середовища: використовувався Vercel, тому потрібно створити в ньому обліковий запит та прив’язати свій GitHub аккаунт.
2. Процес розгортання:
    - Імпорт проекта: Для початку на платформі Vercel нам потрібно імпортувати проект. Натиснувши “New Project”, треба обрати репозиторій у якому зберігається додаток Студабр.
    - Налаштування проекта: треба обрати гілку з якої буде виконано розгортання застосунку. В моєму випадку це - main.
    - Налаштування змінних оточення: це дуже важливий етап, на ньому ми повинні вказати для сайту Студабр посилання на

серверну частину, та посилання на сховище файлів, де ми зберігаємо картинки та маркдаун файли.
Приклад:

```
REACT_APP_API_URL=https://localhost:3000
REACT_APP_DEV_STORAGE_LINK=http://127.0.0.1:10000/devstoreaccount1/files
```

- Після налаштування проекта натискаємо кнопку “Deploy”, Vercel самостійно встановити бібліотеки, які були вказані в “package.json”, та почне вигружати сайт.
- У кінці, після успішного розгортання, ми отримаємо посилання на наш сайт в інтернеті.

ВИСНОВОК

У ході розробки дипломного проекту була реалізована користувацька частина веб-застосунка Студабр. Було обрано ефективні та сучасні інструменти, які забезпечили продуктивність, гнучкість та зручність застосунку. Як результат, була отримана повноцінна платформа, яка задовольняє потреби середньостатистичного користувача, як з функціональної сторони, так і з візуальної.

Основні здобутки:

- Розроблено повноцінний текстовий редактор, з підтримкою форматування, вставки картинок та посилань. Також створено віджети які дозволяють визначати поведінку редактору.
- Реалізована ефективна логіка опрацювання картинок та їх зберігання у сховищі файлів.
- Створено спосіб для написання та зберігання контенту публікацій у форматі markdown. Це дає приріст у швидкості та ефективності роботи застосунку.
- Розроблено логіку для зберігання JWT-token у cookies, та його використання для роботи з захищеною частиною серверу.
- Створено сучасну, зручну та ефективну архітектуру застосунку, яка відповідає більшості сучасних вимог.
- Реалізовано якісну маршрутизацію між сторінками.
- Розроблений сучасний дизайн сайту.
- Використано сучасні бібліотеки та фреймворки при створенні застосунку.
- Проведене якісне тестування функціональної та візуальної частини сайту.
- Виконаний процес вигрузки сайту у мережу.
- Створена ефективна фильтрація та пагінація.

Виклики та їх подолання:

- Стилізація проекту у стилі сучасного мінімалізму: було використано фреймворк Bootstrap, налаштовано тему для сайту, створено стилізовані компоненти.

- Текстовий Редактор: Розроблено спеціальні вузли, компоненти та інша логіка для відображення картинок, посилань та форматування в текстовому редакторі, з використанням markdown розмітки.
- Фільтрація та пагінація: реалізована логіка для швидкої фільтрації та підрахунку кількості публікацій в пагінації на користувацькій стороні, за допомогою методів та функцій наданих мовою TypeScript.
- Оптимізація швидкості відображення сторінок додатку: Було використано функціональні компоненти, ледаче завантаження та Suspense.

Перспективи розвитку:

- Додавання збору дій користувача, для відстежування можливих слабких місць застосунку та створення статистики.
- Створення окремої сторінки профілю користувача.
- Розширення можливостей текстового редактору, наприклад додавання таблиць, відео та коду.
- Інтеграція зі штучним інтелектом, для більш персоналізованого підбору публікацій.
- Створення можливості пов’язувати публікації у курси.
- Реалізація логіки для імпорту та експорту контенту публікації.

Ці вдосконалення зроблять сайт Студабр більш функціональним та зручним. Забезпечать його унікальність та популярність.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React Docs. URL: https://react.dev/learn (дата доступу: 29.05.2024)
2. Getting started with React. URL: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started (дата доступу: 14.03.2024)
3. Компонентно-орієнтована архітектура. URL: https://appmaster.io/ru/blog/komponentnaia-arkhitektura-na-osnove-reaktivnykh-komponentov (дата доступу: 01.06.2024)
4. 8 Best Practices for React Component Design. URL: https://dev.to/blossom/8-best-practices-for-reactjs-component-design-4jn5#:~:text=Here%20are%20some%20best%20practices%20to%20consider%3A%201.Test%20that%20your%20components%20handle%20user%20interactions%20correctly (дата доступу: 17.05.2024)
5. Getting Started with Redux. URL: https://redux.js.org/introduction/getting-started (дата доступу: 28.05.2024)
6. Redux Best Practices. URL: https://dev.to/martinrojas/redux-best-practices-7l4 (дата доступу: 10.04.2024)
7. An introduction to Wouter: A React Router alternative. URL: https://blog.logrocket.com/an-introduction-to-wouter-a-react-router-alternative/ (дата доступу: 20.02.2024)
8. Wouter Docs. URL: https://github.com/molefrog/wouter (дата доступу: 03.06.2024)
9. Axios Docs. URL: https://axios-http.com/docs/intro (дата доступу: 03.06.2024)
10. React Best Practices. URL: https://www.freecodecamp.org/news/best-practices-for-react/ (дата доступу: 04.06.2024)
11. Get started with Bootstrap. URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/ (дата доступу: 14.01.2024)
12. Introduction | Lexical. URL: https://lexical.dev/docs/intro (дата доступу: 18.03.2024)
13. Nodes | Lexical. URL: https://lexical.dev/docs/concepts/nodes (дата доступу: 04.06.2024)

14. React Icons Docs. URL: https://react-icons.github.io/react-icons/ (дата доступу: 06.06.2024)
15. Mocha for TypeScript Testing: How to Get Started. URL: https://www.testim.io/blog/mocha-for-typescript-testing/ (дата доступу: 29.05.2024)
16. Mocha/Chai with TypeScript. URL: https://dev.to/matteobruni/mocha-chai-with-typescript-37f (дата доступу: 01.06.2024)
17. TypeScript: Documentation. URL: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html (дата доступу: 03.06.2024)
18. Best practices that you can apply to your TypeScript project. URL: https://github.com/andredesousa/typescript-best-practices?tab=readme-ov-file#testing (дата доступу: 18.04.2024)
19. How To Build A Text Editor With Lexical and React. URL: https://konstantin.digital/blog/how-to-build-a-text-editor-with-lexical-and-react (дата доступу: 19.05.2024)
20. 9 ways to deploy a React app for free. URL: https://blog.logrocket.com/9-ways-deploy-react-app-free/ (дата доступу: 01.06.2024)
21. Get started with Vercel. URL: https://vercel.com/docs/getting-started-with-vercel (дата доступу: 01.06.2024)
22. Adding Custom Environment Variables. URL: https://create-react-app.dev/docs/adding-custom-environment-variables/ (дата доступу: 01.06.2024)
23. Introduction | React Bootstrap. URL: https://react-bootstrap.github.io/docs/getting-started/introduction (дата доступу: 15.04.2024)
24. Basic Syntax | Markdown Guide. URL: https://www.markdownguide.org/basic-syntax/ (дата доступу: 30.05.2024)
25. Building a rich text editor with Lexical and React. URL: https://blog.logrocket.com/build-rich-text-editor-lexical-react/ (дата доступу: 14.04.2024)
26. Introduction | Testing Library. URL: https://testing-library.com/docs/ (дата доступу: 27.05.2024)

27. React Testing Library. URL: https://testing-library.com/docs/react-testing-library/intro (дата доступу: 27.05.2024)
28. Sample Terms of Use Template and Guide. URL: https://termly.io/resources/templates/terms-of-use-template/ (дата доступу: 01.04.2024)
29. Sample Privacy Policy Template: Free Website Example. URL: https://termly.io/resources/templates/privacy-policy-template/ (дата доступу: 01.04.2024)
30. Modern Frontend Architecture. URL: https://medium.com/js-dojo/modern-frontend-architecture-101-f9c88c20ea20 (дата доступу: 01.06.2024)
31. Що таке інтеграційне тестування?. URL: https://www.guru99.com/uk/integration-testing.html (дата доступу: 02.05.2024)
32. Інтеграційне тестування. URL: https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F (дата доступу: 02.05.2024)
33. Що таке модульне тестування?. URL: https://www.guru99.com/uk/module-testing.html (дата доступу: 03.05.2024)
34. Unit Testing - Software Testing. URL: https://www.geeksforgeeks.org/unit-testing-software-testing/ (дата доступу: 03.05.2024)
35. It’s become harder to find information on the web: Google & the rise of junk articles. URL: https://www.reddit.com/r/google/comments/v7zewn/its_become_harder_to_find_information_on_the_web/ (дата доступу: 01.06.2024)
36. Finding good information on the internet. URL: https://www.scientificamerican.com/blog/guest-blog/finding-good-information-on-the-internet/ (дата доступу: 01.06.2024)
37. Вплив платформ для онлайн-курсів на навчання в школах. URL: https://osvita.ua/news/89291/ (дата доступу: 31.05.2024)
38. Статистика та тенденції онлайн-навчання за 2024 рік - Website Rating. URL:

https://www.websiterating.com/uk/blog/research/online-learning-statistics/ (дата доступу: 01.06.2024)
39. 7 етапів розробки сайту: як правильно розробити сайт. URL: https://brander.ua/blog/kh-etapiv-rozrobki (дата доступу: 05.05.2024)
40. An Ultimate Guide to Front-end Development Process [6 Steps] (monocubed.com). URL: https://www.monocubed.com/blog/front-end-development-process/ (дата доступу: 01.06.2024)
41. Front End Developer – What is Front End Development, Explained in Plain English (freecodecamp.org). URL: https://www.freecodecamp.org/news/front-end-developer-what-is-front-end-development-explained-in-plain-english/ (дата доступу: 02.06.2024)
42. Найкращі практики для створення надійних React-додатків (web-developer.in.ua). URL: https://web-developer.in.ua/assets/articles/react/react-practices/react-practices.html (дата доступу: 01.06.2024)
43. React і чому він такий популярний | by Jstify Community | Medium. URL: https://medium.com/@jstify.community/react-%D1%96-%D1%87%D0%BE%D0%BC%D1%83-%D0%B2%D1%96%D0%BD-%D1%82%D0%B0%D0%BA%D0%B8%D0%B9-%D0%BF%D0%BE%D0%BF%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%B8%D0%B9-766b789c9ea2 (дата доступу: 02.06.2024)
44. Why You Should Use React.js For Web Development (freecodecamp.org). URL: https://www.freecodecamp.org/news/why-use-react-for-web-development/ (дата доступу: 02.06.2024)
45. How to Manipulate Strings in JavaScript – With Code Examples (freecodecamp.org). URL: https://www.freecodecamp.org/news/how-to-manipulate-strings-in-javascript/ (дата доступу: 14.05.2024)
46. JavaScript: изучаем регулярные выражения на практике / Хабр (habr.com). URL: https://habr.com/ru/articles/565726/ (дата доступу: 15.05.2024)
47. How to Use Regex in TypeScript | Delft Stack. URL: https://www.delftstack.com/howto/typescript/typescript-using-regex-in-typescript/ (дата доступу: 14.05.2024)

48. Використання TypeScript – React. URL: https://uk.react.dev/learn/typescript (дата доступу: 01.06.2024)
49. Redux vs Context API: When to use them - DEV Community. URL: https://dev.to/ruppysuppy/redux-vs-context-api-when-to-use-them-4k3p (дата доступу: 03.06.2024)
50. Tutorial v6.23.1 | React Router. URL: https://reactrouter.com/en/main/start/tutorial (дата доступу: 02.06.2024)
51. Webpack: встановлення та налаштування, лоадери та плагіни (foxminded.ua). URL: https://foxminded.ua/webpack/ (дата доступу: 01.06.2024)
52. Getting Started | webpack. URL: https://webpack.js.org/guides/getting-started/ (дата доступу: 01.06.2024)
53. Поліфіли та транспілятори (javascript.info). URL: https://uk.javascript.info/polyfills (дата доступу: 28.05.2024)
54. Folder Structure | Create React App (create-react-app.dev). URL: https://create-react-app.dev/docs/folder-structure (дата доступу: 21.05.2024)
55. Getting Started | Create React App (create-react-app.dev). URL: https://create-react-app.dev/docs/getting-started (дата доступу: 21.05.2024)
56. Configuring Your Store | Redux. URL: https://redux.js.org/usage/configuring-your-store (дата доступу: 02.06.2024)
57. Request Config | Axios Docs (axios-http.com). URL: https://axios-http.com/docs/req_config (дата доступу: 03.06.2024)
58. Розширення та плагіни для Visual Studio Code. Топ 10 найкращих (itstep.org). URL: https://cloud.itstep.org/blog_3/top-10-best-vs-extensions-and-useful-plugins-for-visual-studio (дата доступу: 08.03.2024)
59. 15 Best VSCode Extensions for Front-End Developers in 2024 (webreaper.dev). URL: https://webreaper.dev/posts/best-vscode-extensions-front-end-developers/ (дата доступу: 08.03.2024)
60. React JS ReactDOM - GeeksforGeeks. URL: https://www.geeksforgeeks.org/reactjs-reactdom/ (дата доступу: 31.05.2024)
61. ReactDOM – React (reactjs.org). URL: https://legacy.reactjs.org/docs/react-dom.html (дата доступу: 31.05.2024)

62. Інструкція Git для новачків: що це таке, як він працює та які є основні команди - DAN.IT (dan-it.com.ua). URL: https://dan-it.com.ua/uk/blog/instrukcija-git-dlja-novachkiv-shho-ce-take-jak-vin-pracjuie-ta-jaki-ie-osnovni-komandi/ (дата доступу: 03.01.2024)
63. About GitHub and Git - GitHub Docs. URL: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git (дата доступу: 03.01.2024)
64. 9 принципів хорошого дизайну сайту | Веб-студія Laweb. URL: https://laweb.com.ua/uk/blog/9-principiv-khoroshogo-dizaynu-saytu (дата доступу: 23.05.2024)
65. Prettier · Opinionated Code Formatter. URL: https://prettier.io/ (дата доступу: 17.05.2024)
66. What is Prettier? · Prettier. URL: https://prettier.io/docs/en/ (дата доступу: 17.05.2024)
67. Why Prettier? · Prettier. URL: https://prettier.io/docs/en/why-prettier (дата доступу: 17.05.2024)
68. Configuration File · Prettier. URL: https://prettier.io/docs/en/configuration (дата доступу: 17.05.2024)
69. Find and fix problems in your JavaScript code - ESLint - Pluggable JavaScript Linter. URL: https://eslint.org/ (дата доступу: 17.05.2024)
70. HTML CSS Support. URL: https://marketplace.visualstudio.com/items?itemName=ecmel.vscode-html-css (дата доступу: 01.04.2024)
71. React Markdown. URL: https://www.npmjs.com/package/react-markdown (дата доступу: 04.06.2024)