Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

другий (магістерський)
(рівень вищої освіти)

на тему: ***Розроблення інтелектуальної системи створення шаблонів документів підприємця***

Виконав: студент VI курсу групи КН
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Воробель Б.І.
(прізвище та ініціали)

Керівник Бекас Б.О., Борецька І.Б.
(прізвище та ініціали)

Рецензент
(прізвище та ініціали)

Львів – 2024

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Борецька І.Б.
"____" _____2024 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Воробель Богдан Ігорович
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення інтелектуальної системи створення шаблонів документів підприємця
Керівник роботи *Бекас Б.О., Борецька І.Б*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "13" лютого 2023 року № С-49
2. Термін подання студентом роботи "05" січня 2024 року
3. Вихідні дані до роботи Розробити інтелектуальну систему для документообігу використовуючи фреймворк React.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Стан проблемної області
Інформаційне забезпечення
Математичне забезпечення
Програмне забезпечення
Розроблення стартап проекту
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка матеріалу до доповіді

6. Дата видачі завдання "15" лютого 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів.	20.02.23-20.04.23	Виконано
2.	Постановка задачі і її формалізація	20.04.23-10.05.23	Виконано
3.	Виконання вхідного етапу технології	10.05.23-25.07.23	Виконано
4.	Реалізація головних алгоритмів проекту	25.07.23-12.09.23	Виконано
5.	Виконання етапу відлагодження проєкту	12.09.23-22.10.23	Виконано
6.	Виконання етапу впровадження та випуску бета-версії.	22.10.23-05.11.23	Виконано
7.	Оформлення записки до дипломного проекту.	05.11.23-05.01.24	Виконано

Студент _____ Воробель Б.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Бекас Б.О., Борецька І.Б
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 74 сторінки пояснювальної записки, 41 рисунок, 6 джерел.

Дипломна робота присвячена розробці застосунку AiDocs, який відкриває нові можливості для швидкого створення різноманітної документації, надаючи зручність і високий функціонал користувачам. AiDocs є потужною платформою для продуктивної роботи з документацією.

Для реалізації поставленої цілі за основу береться JavaScript бібліотека React та кросплатформенне середовище Node.js

Ключові слова: JavaScript, Node.js, React, програмне забезпечення, документ

ABSTRACT

The thesis contains 74 pages of explanatory note, 41 figures, 6 sources.

The thesis is devoted to the development of the AiDocs application, which opens up new opportunities for the rapid creation of various documentation, providing convenience and high functionality to users. AiDocs is a powerful platform for productive work with documentation.

To realize this goal, the JavaScript library React and the cross-platform environment Node.js are taken as a basis.

Keywords: JavaScript, Node.js, React, software, document

ТЕХНІЧНЕ ЗАВДАННЯ

У відповідності з вимогами технічного завдання розробити інтелектуальну систему створення шаблонів документів підриємств. Користувачі можуть швидко створювати нові шаблони на основі існуючих. Впровадити можливість додавання різних елементів та полів для персоналізації шаблонів. Створити базу шаблонів, де користувачі можуть вибирати необхідні опції для своїх документів. Також, передбачити можливість налаштовувати обраних шаблонів для відповідності конкретним потребам користувача. Врахувати опцію із завантаження документу у форматі PDF. Забезпечення зручності та доступності використання для кінцевих користувачів.

ПЕРЕЛІК СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Кросплатформенна візуалізація (КВ) – це такий вид комп'ютерної віртуалізації, що дозволяє програмному продукту, що був скомпільований на одному комп'ютері (з одним апаратним забезпеченням та операційною системою), бути перенесеним без змін на інший комп'ютер з іншим апаратним та/або програмним забезпечення..

ПЗ – Програмне забезпечення.

Фреймворк (Framework), каркас, платформа, структура, інфраструктура — інфраструктура програмних рішень, що полегшує розробку складних систем. Спрощено дану інфраструктуру можна вважати своєрідною комплексною бібліотекою, але при цьому вона має ряд обмежень, що задають правила створення структури проєкту та написання коду.

ЗМІСТ

ВСТУП

AiDocs - інноваційна система для створення та управління документами, побудована на фреймворку React, володіє вражаючим функціоналом. Можливість швидко створювати нові шаблони на основі наявних - це чудовий спосіб зробити процес створення документів більш ефективним та продуктивним.

База шаблонів дозволяє користувачам вибирати ідеальні варіанти для своїх потреб і зручно їх налаштовувати. Можливість роздрукувати або завантажити створені документи у форматі PDF - це додатковий плюс для зручності користувачів.

Об’єктом дослідження є система управління документами AiDocs

Метою роботи є розробка системи керування документами для подальшого ефективного використання та для розвитку системи.

Предметом дослідження є реалізація основних підходів в програмуванні з використанням React та середовища Node.js

Практичне значення у зручності інтерфейсу, можливості завантажувати та використовувати доступні на шаблони і матеріали.

Наукова новизна попередньо запрограмованого контролера пришвидшить запуск верстата в роботу.

Актуальність Системи електронного документообігу відіграють важливу роль у модернізації та удосконаленні діловодних процесів як у державних, так і в комерційних установах. Вони сприяють стандартизації та упорядкуванню процесів управління документами, роблять доступ до них зручним та ефективним, а також забезпечують автоматизацію багатьох рутинних операцій.

Системи AiDocs дозволяє впорядковувати потік документів, зберігати, обробляти та відстежувати їхню обробку в електронному вигляді, спрощує комунікацію між різними відділами та працівниками, забезпечуючи швидкий обмін інформацією та сприяючи оперативній роботі колективу. Такі системи не лише економлять час, а й покращують загальну ефективність роботи, роблячи робочий процес більш прозорим, точним та доступним для усіх зацікавлених сторін.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Огляд проблемної області.

Системи електронного документообігу використовуються в різних сферах бізнесу та й, навіть, особистого життя. Зокрема, системи електронного документообігу відіграють ключову роль у структуруванні діловодних процесів у державних та комерційних установах; приведення їх до єдиного порядку, а також оптимізації роботи працівників, забезпечуючи ефективний і безперешкодний доступ до документів з функцією автоматизації рутинних операцій з відстеження і пошуку необхідної інформації та формування звітів про документообіг.

Однак, щороку утворюється дуже великий обсяг документів з регламентованим часом їх опрацювання, а якість та оперативність взаємодії з документами значною мірою визначають ефективність та результативність роботи. З розвитком електронного урядування кількість оброблюваних запитів може сягати кількох тисяч на день. Використання системи управління документами на базі фреймворку React надає користувачам можливість легко генерувати нові шаблони на основі існуючих, що робить процес створення документів швидким та ефективним. Це прискорює обробку документів, готує дані, необхідні для прийняття рішень людиною, а також запобігає людським помилкам.

Історія управління документами кардинально змінилася у 1980-х роках зі зростанням доступності комп'ютерних технологій. Розвиток серверів дозволив організаціям зберігати документи в електронному вигляді в централізованих мейнфреймах. Це стало початком систем електронного документообігу. Тим часом, винахід сканерів уможливив перетворення паперових документів на цифрові документи в цифрову форму. Розвиток комп'ютерів дозволив компаніям створювати та зберігати документи на комп'ютерах в офісі. Сучасні системи електронного документообігу створюють можливість централізовано зберігати великі обсяги цифрових документів. Для того, щоб забезпечити хорошу класифікацію електронних документів, багато систем електронного документообігу покладаються на детальний процес зберігання документів, що включає певні

елементи, які називаються метаданими. Саме це робить систему електронного документообігу такою цінною для бізнесу чи організації.

Будь-яка діяльність відображається в документах, будь то управління, фінанси чи виробництво, тому документообіг є життєво важливою системою для організації. Досить часто автоматизація документообігу може суттєво покращити бізнес-процеси всієї організації. Масове збільшення обсягів електронного документообігу у зв'язку з пандемією призвело до збільшення механічної однотипної роботи працівників, керівників та працівників служб документообігу організацій, які реєструють та відповідають на тисячу і більше документів на день. Це призвело до збільшення трудових і часових витрат. Актуальність цієї проблеми в сучасних умовах полягає в удосконаленні традиційних систем електронного документообігу, шляхом застосування методів аналізу даних та машинного навчання, для оптимізації роботи співробітників організації та якісного проходження всього життєвого циклу електронного документа з мінімальним втручанням людини в процес.

Створені технології та інструментальні засоби забезпечать моделювання процесів вилучення інформації та створення систем автоматичного опрацювання текстів на основі онтології предметної області та лінгвістичних знань, представлених у вигляді предметної області та лінгвістичних знань, представлених предметними словниками та фактографічними моделями.

1.2. Можливості й переваги застосунку " AiDocs ".

AiDocs - це інноваційна система для створення та управління документами, розроблена на базі фреймворку React. Цей інтелектуальний інструмент надає користувачам можливість легко генерувати нові шаблони на основі існуючих, що робить процес створення документів швидким та ефективним.

Завдяки великій базі шаблонів AiDocs користувачі можуть вибирати ідеальний шаблон для своїх потреб та легко його налаштовувати. Система надає можливість роздрукувати створені документи або завантажити їх у форматі PDF. Однак ці функції доступні лише для зареєстрованих користувачів, що додає додатковий рівень безпеки та конфіденційності.

Для незареєстрованих користувачів AiDocs надає можливість переглядати захищені водяними знаками зображення шаблонів документів. Це дозволяє користувачам оцінити функціональність та різноманіття програми перед реєстрацією.

Використання React у розробці AiDocs гарантує високу реактивність та зручний інтерфейс для користувачів. Компонентний підхід дозволяє легко взаємодіяти з різними функціями системи та ефективно користуватися її можливостями.

1.3. Висновки до розділу

Узагальнюючи, Ai Docs є інноваційним інструментом, що допомагає легко та швидко створювати різноманітну документацію, забезпечуючи високий рівень зручності та функціональності для своїх користувачів.AiDocs - це не просто інструмент для створення документів, але і потужна платформа з усіма необхідними інструментами для продуктивної роботи з документацією.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. React — відкрита JavaScript бібліотека.

Однією з ключових переваг використання React в AiDocs є використання віртуального DOM, що призводить до швидкого та ефективного відображення змін в інтерфейсі. Цей підхід допомагає підтримувати високу продуктивність додатку та зменшує навантаження на браузер користувача. AiDocs використовує принцип одностороннього потоку даних для ефективного відстеження та керування станом компонентів. Це забезпечує послідовність та передбачуваність роботи, а також полегшує виявлення та виправлення помилок.

Згідно з його офіційним слоганом, React - це бібліотека для створення користувацьких інтерфейсів. React не є фреймворком - він навіть не є ексклюзивним для Інтернету. Він використовується з іншими бібліотеками для рендерингу в певних середовищах. Наприклад, React Native можна використовувати для створення мобільних додатків.

Для створення веб-додатків розробники використовують React у тандемі з ReactDOM. React та ReactDOM часто обговорюються в тих же просторах, що й інші фреймворки для веб-розробки, і використовуються для вирішення тих же проблем, що й інші фреймворки для веб-розробки. Коли ми говоримо про React як про "фреймворк", ми маємо на увазі саме це розмовне розуміння.

Основна мета React - мінімізувати помилки, які виникають під час створення користувацького інтерфейсу. Це досягається завдяки використанню компонентів - самодостатніх, логічних фрагментів коду, які описують частину інтерфейсу користувача. Ці компоненти можна компонувати разом, щоб створити повноцінний інтерфейс, і React абстрагує більшу частину роботи з рендерингу, залишаючи вам можливість сконцентруватися на дизайні інтерфейсу.

На відміну від інших фреймворків, що розглядаються в цьому модулі, React не запроваджує суворих правил щодо конвенцій коду або організації файлів. Це дозволяє командам встановлювати конвенції, які працюють найкраще для них, і адаптувати React у будь-який спосіб, який вони бажають. React може працювати з однією кнопкою, кількома частинами інтерфейсу або з усім користувацьким

інтерфейсом додатку.

В епоху висококонкурентної розробки мобільних та веб-додатків попит на надійні та ефективні технології розробки постійно зростає. React.js, яка зарекомендувала себе як одна з найпродуктивніших та найпопулярніших інтерфейсних JavaScript-бібліотек, знаходиться в авангарді цього цифрового ландшафту, що постійно розвивається. Вона швидко стає основною бібліотекою для розробників, оскільки надає численні переваги для створення зручних та інтерактивних веб-додатків.

React.js - популярна JavaScript бібліотека для створення користувацьких інтерфейсів. Вона широко використовується для розробки односторінкових додатків (SPA) та інтерактивних веб-інтерфейсів.

Компонентно-орієнтована архітектура: React.js використовує компонентну архітектуру, де додатки створюються шляхом компонування багаторазових та модульних компонентів.

На практичному прикладі це виглядає так:

Уявімо, що користувач ставить лайк під вподобаною фотографією. Кількість лайків збільшилася одразу, як ви вподобали фотографію. Без жодного перезавантаження сторінки, просто якимось магічним чином кількість лайків змінилася. Ця магія, це react.js.

Чудовим прикладом використання react.js є Facebook. Мільярди постів, мільйони лайків. Як вдається обробляти все це з такою шаленою швидкістю. Для цього створено бібліотеку під назвою react.js

Кожен пост в Facebook є контейнером та складається з декількох маленьких частин, таких як лайки, коментарі, поділитися, поле для коментарів, які називаються компонентами. Так, за допомогою react сторінку можна розбити на компоненти. Весь веб базується на html. Все почалося з простого HTML-коду, і люди були в захваті від нього в 90-х роках. Потім з'явився Javascript у 1996 році з можливостями взаємодії з HTML DOM (Document Object Model - модель об'єктів документа). Після цього з'явився Jquery для зміни вмісту HTML DOM. Наступним після цього в 2009 році з'явився Angular.js як батько фронтенд-фреймворків, який

дав повноцінну можливість створити сильний фронтенд. Зараз Javascript набагато потужніший за HTML, Facebook врахував цей факт і вирішив створити сам HTML з Javascript Отже, підводячи підсумок, react.js створює html з javascript. Але сам react js не написаний на звичайному джаваскрипті. Для цього використовується трансполятор, який називається Babel. Babel перетворює JSX в JS, який можна вставити в html для динамічного створення контенту.

У React є поняття віртуального DOM. Скажімо, html має згенерований вихідний javascript, який створює html. Тепер припустимо, що коли щось змінюється в певному компоненті, змінюється і цей компонент. Тепер ці зміни не передаються безпосередньо в DOM, а передаються в щось, що називається VirtualDOM. У цей момент виникає різниця між VirtualDOM і реальним HTML DOM, і зміни переносяться в реальний DOM. Це те, що робить його швидким.

2.2. Остання версія React.js та її особливості

Остання версія популярної бібліотеки JavaScript, що пропонує нові цікаві функції та покращення. React 18 - це значний крок вперед в еволюції цієї бібліотеки JavaScript. У ній представлено кілька нових функцій та вдосконалень, які мають на меті покращити досвід розробників та продуктивність React-додатків. Щоб забезпечити плавний перехід на React 18, дуже важливо мати чітке розуміння того, що ця версія привносить в роботу:

1. Паралельний режим

React 18 представляє паралельний режим, революційну функцію, яка змінює те, як React працює з рендерингом. У попередніх версіях React оновлення рендерингу були синхронними, що могло призводити до вузьких місць у роботі додатків зі складним користувацьким інтерфейсом.

Паралельний режим змінює це, вмикаючи асинхронний рендеринг. Це дозволяє React працювати над декількома завданнями одночасно, роблячи ваш додаток більш чуйним та ефективним. У паралельному режимі React може визначати пріоритети та призупиняти роботу рендерингу, гарантуючи, що найважливіші частини вашого додатку завжди реагують на взаємодію з

користувачем. Це особливо корисно для додатків з високим рівнем інтерактивності та динамічним контентом.

2. Автоматичне пакетування

У React 18 автоматичне групування є ще одним ключовим покращенням, яке спрощує роботу з оновленнями станів. У попередніх версіях React часто доводилося вручну групувати оновлення станів, щоб мінімізувати проблеми з продуктивністю. Завдяки автоматичному пакетуванню React розумно групує оновлення станів, зменшуючи потребу розробників вручну оптимізувати продуктивність рендерингу.

Це означає, що ви можете писати чистіший та зручніший для супроводу код без необхідності мікро-оптимізації оновлень компонентів. React автоматично групує зміни станів і застосовує їх в оптимальний спосіб, що робить роботу розробника більш комфортною.

3. Призупинення для отримання даних

Отримання даних у React було спрощено завдяки введенню Suspense для отримання даних. У React 18 ви можете використовувати Suspense для декларативної обробки асинхронного завантаження даних та керування станами завантаження. Це полегшує координацію завантаження даних у ваших компонентах і забезпечує кращий користувацький досвід.

За допомогою Suspense ви можете вказати, які частини вашого дерева компонентів мають чекати на завантаження даних, а React автоматично керуватиме станами завантаження та помилок. Ця функція спрощує роботу з асинхронними операціями, зменшуючи потребу в складному коді для управління завантаженням і сценаріями помилок.

4. Межі помилок

Обробка помилок у React-додатках стала більш надійною завдяки покращеним межам помилок у React 18, що є важливим аспектом для тих, хто мігрує на React 18. Error Boundaries - це спеціальні компоненти, які перехоплюють помилки у своїх дочірніх компонентах і дозволяють витончено обробляти їх, не спричиняючи повного падіння додатку. React 18 покращує межі помилок,

надаючи більше гнучкості та контролю над обробкою помилок.

Це допомагає ізолювати помилки та відновлюватися після них, гарантуючи, що ваш додаток залишається стабільним та швидко реагує навіть тоді, коли виникають проблеми. З React 18 ви можете визначити власні межі помилок і вказати, як обробляти помилки у спосіб, що найкраще відповідає потребам вашого додатку.

Нові клієнтські та серверні API рендерингу дозволяють користувачам продовжувати використовувати старі API в режимі React 17 при переході на нові API в React 18. Це полегшує розробку високопродуктивних React-додатків. Ці API призначені для спільної роботи та підвищення продуктивності додатку як на клієнті, так і на сервері.

2.3. Node.js кросплатформенне середовище

Node.js, кросплатформенне середовище виконання JavaScript з відкритим вихідним кодом, створене на основі движка Chrome V8, руйнує архаїчні обмеження, які колись утримували JavaScript в межах браузера. Його новаторські можливості прокладають шлях до високопродуктивних, масштабованих веб-додатків. Розуміння переваг Node.js та можливостей його використання є життєво важливим, але для цього необхідно попередньо розібратися в його архітектурі та механіці роботи.

Архітектура Node.js базується на двох ключових принципах: неблокування операцій вводу/виводу та асинхронна парадигма.

Перший принцип, неблокування операцій вводу/виводу, показує, як Node.js використовує ці операції для полегшення безперебійної обробки клієнтських запитів. Ця методологія дозволяє одночасне виконання операцій вводу/виводу без перешкод для основного потоку. Перевага такого підходу подвійна: він помітно підвищує ефективність, забезпечуючи при цьому вищий ступінь реагування.

Другий принцип, асинхронна парадигма, ілюструє прихильність Node.js до асинхронної моделі програмування. Це легко узгоджується з подієво-керованою сутністю JavaScript. Завдяки розумному використанню зворотних викликів та

подієво-керованого програмування, Node.js оптимізує використання ресурсів.

Node.js швидко перетворився на фундаментальну основу сучасної веб-розробки. Його стрімке зростання можна пояснити значною мірою широким спектром переваг, які він надає користувачам. Ключові з них:

1. Масштабованість та продуктивність

Node.js був ретельно розроблений з постійним фокусом на масштабованість та продуктивність. Його революційна парадигма, керована подіями та неблокуванням, легко організовує швидку та кваліфіковану обробку безлічі одночасних запитів. Крім того, потужний Google V8, на якому працює Node.js, старанно компілює JavaScript у машинний код, що сприяє блискавичному виконанню.

2. Універсальність та використання JavaScript

Основною перевагою Node.js є підтримка серверного JavaScript, що об'єднує клієнтську і серверну розробку, спрощує процес і прискорює цикли розробки. Широке використання JavaScript ще більше підвищує привабливість Node.js завдяки великій кількості ресурсів, навчальних посібників та підтримці спільноти.

3. Повторне використання та спільне використання коду

Node.js заохочує повторне використання та поширення коду. Завдяки npm розробники можуть упаковувати свої рішення в модулі та ділитися ними зі спільнотою або імпортувати пакети, створені іншими, щоб не "винаходити велосипед". Цей принцип багаторазового використання коду може значно скоротити цикли розробки та сприяти створенню підтримуваного та чистого коду.

4. Багата екосистема

Завдяки npm, Node.js може похвалитися найбільшою у світі екосистемою бібліотек з відкритим вихідним кодом, що надає безліч готових рішень і пакетів, які можуть значно прискорити процес розробки. Ця динамічна екосистема сприяє постійним інноваціям та співпраці, додаючи елемент підтримки спільноти, який цінують розробники по всьому світу.

5. Архітектура мікросервісів

Нарешті, Node.js панує як найкращий вибір для створення додатків в архітектурі мікросервісів. Його легка природа та модульний дизайн дозволяють легко розбивати складні додатки на прості, відокремлені та керовані мікросервіси. Це може значно покращити гнучкість, відмовостійкість та масштабованість додатків.

Підсумовуючи, дуже важливо сприймати Node.js як щось більше, ніж просто середовище виконання. Це всеохоплююче рішення, пристосоване до динамічних потреб сучасної веб-розробки.

2.4. Висновки до розділу

Завдяки автоматичному пакетуванню, призупиненню та інтеграції з перехідним API, розробники отримали надійні інструменти для створення більш чутливих, ефективних та інтерактивних користувацьких інтерфейсів. Оскільки React продовжує розвиватися, він стає незамінним фреймворком для створення сучасних веб-додатків.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Основні відомості про реляційні бази даних

Бази даних стали невід'ємною складовою багатьох сфер професійної діяльності. Вони допомагають збирати, зберігати, організовувати та надавати доступ до великого обсягу інформації. Системи управління базами даних (СУБД) дозволяють структурувати цю інформацію, забезпечуючи швидкий і ефективний доступ до неї. СУБД використовуються в бізнесі, науці, медицині, освіті та багатьох інших галузях. Вони дозволяють зберігати великі обсяги даних у відформатованому вигляді, забезпечують можливість швидкої обробки та аналізу цих даних. Це допомагає приймати обгрунтовані рішення, прогнозувати тенденції та вдосконалювати процеси в різних сферах діяльності.

Організаційний метод, що використовується для підтримки баз даних, дуже важливий. Він визначає структуру, спосіб доступу до даних, методи захисту інформації та багато іншого. Ефективне управління базами даних допомагає забезпечити їхню надійність, швидкість та гнучкість у використанні, що є ключовим для багатьох організацій.

Системи управління базами даних мають довгу історію розвитку, і їхні можливості продовжують розширюватися з відповіддю на зростаючі вимоги бізнесу та збільшення обсягів даних. Початково бази даних створювалися для зберігання і організації даних на великих ЕОМ. З тих пір вони пройшли значний шлях, стали більш ефективними, гнучкими та потужними завдяки постійному розвитку технологій. Розвиток Інтернету та поширення цифрових технологій значно змінили підхід до управління базами даних. Зараз важливо не лише зберігати і організовувати інформацію, але й забезпечити швидкий доступ до неї, забезпечити безпеку та здатність працювати з великими обсягами даних, що надходять з різних джерел.

Останні тенденції у сфері систем управління базами даних включають в себе розвиток хмарних технологій, аналіз Big Data, використання штучного інтелекту та машинного навчання для оптимізації обробки даних. Ці напрями дозволяють бізнесу отримувати більше інсайтів з даних та працювати з ними більш ефективно

для досягнення своїх цілей.

Реляційні бази даних

Системи управління реляційними базами даних (СУРБД) грають важливу роль у керуванні основними потоками інформації у сучасних підприємствах. Ці системи базуються на концепції реляційної моделі даних, яка використовує таблиці зі зв'язками між ними для зберігання даних. Це дозволяє ефективно організовувати та управляти великими обсягами інформації.

Об'єднання реляційних баз даних і клієнт-серверних технологій дає підприємствам можливість краще керувати своїми даними. Клієнт-серверна архітектура дозволяє використовувати розподілену систему, де сервери зберігають бази даних, а клієнти мають доступ до цих даних через мережу. Це сприяє швидкому доступу до інформації, безпеці та забезпечує можливість одночасної роботи з даними багатьма користувачами.

Управління даними є критичним аспектом для конкурентоспроможності бізнесу. СУРБД надають зручні інструменти для зберігання, організації та доступу до даних, що дозволяє підприємствам ефективно використовувати свої ресурси та приймати обґрунтовані управлінські рішення на основі аналізу цих даних.

Реляційні бази даних (РБД) ґрунтуються на математичній теорії відносин, що дозволяє їм ефективно управляти даними за допомогою мов запитів.

Мови алгебри та числення предикатів є ключовими засобами для виразу запитів до реляційних баз даних. Алгебра реляційних баз даних дозволяє використовувати спеціалізовані оператори для операцій над відношеннями, тоді як числення предикатів визначає нові відношення з використанням правил, які описують умови вибору та об'єднання даних з існуючих відношень.

У реляційній моделі дані представлені у вигляді таблиць або відношень, де кожен рядок відповідає конкретному екземпляру об'єкту, а кожен стовпчик відображає його атрибути. Ця структура дозволяє ефективно зберігати та організовувати дані.

Первинний ключ у таблиці бази даних є унікальним ідентифікатором для кожного запису. Це поле або набір полів, які ідентифікують кожен екземпляр

об'єкту. Він має бути унікальним і не може містити дублікатів або значень, що повторюються, а також він повинен бути мінімально достатнім для цієї цілі. Тобто його значення точно визначає одиничний запис в базі даних.

Реляційні зв'язки між таблицями баз даних

Зв'язки між об'єктами реального світу можуть бути відображені в базі даних через відносини або зв'язки між таблицями. Це може бути відношення батьківської таблиці до підлеглої, де кожен запис в батьківській таблиці може мати один або декілька відповідних записів в підлеглій таблиці.

Це відображення відносин між об'єктами реального світу називається зв'язком бази даних. Наприклад, у випадку бази даних для управління компанією, зв'язок батьківської таблиці "відділи" може мати зв'язок до підлеглої таблиці "співробітники", що вказує на те, що кожен відділ може мати декілька співробітників, але кожен співробітник прив'язаний до конкретного відділу.

Це важливий аспект в реляційних базах даних, оскільки він дозволяє моделювати складні структури даних і відображати зв'язки між різними об'єктами чи сутностями у реальному світі, що допомагає у кращому розумінні та обробці інформації.

3.2. Система управління базами даних MySQL

MySQL є однією з найпопулярніших систем управління базами даних (СУБД), вона відома своєю швидкодією, простотою використання та відкритим кодом. Дана СУБД була спочатку створена як альтернатива комерційним системам, і з часом стала популярним рішенням для веб-розробників, оскільки має відмінну підтримку різних мов програмування.

MySQL розвивався з моменту свого запуску, стаючи більш функціональним та розширеним. Вона спочатку була схожою на mSQL, але з часом вона стала більш потужною та гнучкою системою з багатьма можливостями.

Особливості MySQL включають багатопотоковий сервер баз даних, який забезпечує високу швидкодію та працездатність навіть при обробці великих обсягів даних. Вона може бути використана для різноманітних малих і середніх за розміром

додатків та має підтримку на різних платформах.

Особливо важливою є її ефективність в UNIX-подібних системах, де підтримка багатопотоковості сприяє підвищенню продуктивності системи. MySQL широко використовується для створення динамічних веб-сторінок, систем управління контентом, інтернет-магазинів та багатьох інших веб-застосунків.

Так, MySQL дійсно має безкоштовну ліцензію для некомерційного використання, що робить її популярним вибором для багатьох користувачів та розробників. Однак, вона має як позитивні, так і негативні аспекти.

Позитивні сторони включають:

- Простоту встановлення та використання, що робить її доступною для широкого кола користувачів.
- Підтримка необмеженої кількості одночасних користувачів бази даних.
- Можливість обробки великої кількості рядків у таблицях (до 50 мільйонів).
- Швидкість виконання команд у базі даних.
- Проста та ефективна система безпеки.

Але також є й деякі обмеження:

- Неіснування підтримки транзакцій, хоча можливо використовувати LOCK/UNLOCK TABLE.
- Відсутність підтримки зовнішніх ключів, що може ускладнити забезпечення цілісності даних.
- Відсутність підтримки тригерів, збережених процедур та представлень, що ускладнює розробку складних та розширених структур даних.

Ці обмеження можуть бути критичними для більш складних проектів або додатків, які вимагають великої різноманітності функцій та підтримки складних взаємозв'язків між даними. Тим не менш, для простих або невеликих проектів MySQL може бути дуже ефективним і зручним рішенням.

У деяких випадках недоліки MySQL, такі як відсутність підтримки транзакцій, зовнішніх ключів, тригерів і збережених процедур, не є критичними для розробки малих і середніх інформаційних систем для робочих груп. Особливо в умовах невеликих проектів або коли система використовується для робочих груп чи

невеликих команд, ці обмеження можуть бути прийнятними.

У таких випадках, де важливо швидко розгорнути базу даних та отримати швидкий доступ до неї, MySQL може бути зручним та ефективним вибором. Простота використання, швидкодія та доступність безкоштовної ліцензії роблять її привабливим варіантом для невеликих проектів.

Але для більш складних систем з високими вимогами до цілісності даних, безпеки та складних операцій з базою даних, ці недоліки можуть стати більш критичними. У таких випадках, розробники можуть розглядати інші СУБД, які мають більш широкі функціональні можливості.

3.3 Алгоритмічне забезпечення

Операція JOIN в SQL використовується для об'єднання даних з двох або більше таблиць у реляційній базі даних. Вона дозволяє поєднувати дані з різних таблиць на основі спільних полях або умов.

Існує декілька типів операцій JOIN у SQL:

1. **INNER JOIN:** Повертає рядки, які мають спільні значення у обох таблицях.
2. **LEFT JOIN (или LEFT OUTER JOIN):** Повертає всі рядки з лівої таблиці і співпадаючі рядки з правої таблиці. Якщо значення у правій таблиці відсутнє, результат буде NULL.
3. **RIGHT JOIN (або RIGHT OUTER JOIN):** Повертає всі рядки з правої таблиці і співпадаючі рядки з лівої таблиці. Якщо значення у лівій таблиці відсутнє, результат буде NULL.
4. **FULL JOIN (або FULL OUTER JOIN):** Повертає рядки, які мають співпадіння у обох таблицях, а також рядки, які відсутні в одній або обох таблицях.
5. **CROSS JOIN:** Повертає декартовий добуток двох таблиць, тобто кожен рядок першої таблиці комбінується з кожним рядком другої таблиці.

JOIN-операції є потужним інструментом для отримання необхідних даних з різних таблиць бази даних, дозволяючи виконувати складні запити та отримувати результати на основі умов та відносин між таблицями.

Звичайно, INNER JOIN є важливою операцією для об'єднання таблиць у базі

даних. Він дозволяє отримати лише ті записи, які мають відповідність у обох таблицях, що об'єднуються.

Щодо використання ON та USING для об'єднання таблиць в SQL:

- **ON:** Використовується для вказівки умови об'єднання на основі стовпців, які можуть мати різні імена в об'єднуваних таблицях.
- **USING:** Використовується тоді, коли об'єднувані стовпці мають однакові назви в обох таблицях.

Щодо CROSS JOIN, це операція об'єднання, яка фактично є еквівалентом INNER JOIN, але без умови об'єднання. CROSS JOIN об'єднує кожен рядок однієї таблиці з кожним рядком іншої таблиці, утворюючи декартовий добуток обох таблиць.

Іншими словами, INNER JOIN та CROSS JOIN можуть бути схожими, але INNER JOIN використовується з умовою об'єднання, тоді як CROSS JOIN не має цієї умови та об'єднує кожен рядок однієї таблиці з кожним рядком іншої таблиці.

Щодо використання умови об'єднання в WHERE, це теж можливо, але використання ON чи USING для визначення умов об'єднання більш чітке й зрозуміле у багатьох випадках, особливо при обробці складних запитів чи багатої таблиці даних.

LEFT JOIN або LEFT OUTER JOIN. Обидва вони працюють однаково і повертають всі записи з лівої таблиці (TableA) і відповідні записи з правої таблиці (TableB), якщо вони існують. Якщо відповідних записів у правій таблиці не знайдено, значення буде NULL.

Зазвичай ключове слово OUTER в LEFT OUTER JOIN може опускатися, бо LEFT JOIN і LEFT OUTER JOIN означають одне й те ж саме. Основна різниця між INNER JOIN та LEFT JOIN полягає у тому, що INNER JOIN виводить лише спільні рядки з обох таблиць, тоді як LEFT JOIN виводить всі рядки з лівої таблиці, навіть якщо вони не мають відповідних записів у правій таблиці.

Запит:

```
SELECT * FROM `TableA`
LEFT JOIN `TableB`
ON `TableA`.`name` = `TableB`.`name`
```

У цьому запиті LEFT JOIN повертає всі записи з TableA, включаючи ті, які не мають відповідних записів у TableB (що в результаті мають значення NULL у відповідних полях TableB). Потім у WHERE ми вказуємо, що ми хочемо побачити лише ті записи, де TableB.id є NULL або 0, що означає, що вони відсутні в TableB або мають значення 0.

Запит:

```
SELECT * FROM `TableA`
LEFT JOIN `TableB`
ON `TableA`.`name` = `TableB`.`name`
WHERE `TableB`.`id` IS NULL
```

RIGHT JOIN працює подібно до LEFT JOIN, але розташування таблиць для з'єднання змінюється. У випадку RIGHT JOIN, таблиця, яка знаходиться справа в запиті, буде визначальною, і всі записи з цієї таблиці будуть включені у результат, навіть якщо вони не мають відповідних записів у лівій таблиці.

Подібно до LEFT JOIN, ключове слово OUTER в RIGHT OUTER JOIN може бути опущене, оскільки RIGHT JOIN і RIGHT OUTER JOIN означають одне й те саме.

NATURAL JOIN - це тип об'єднання таблиць в SQL, де умови об'єднання визначаються автоматично на основі стовпців з однаковими іменами в об'єднуваних таблицях. Однак, цей тип об'єднання має свої недоліки і ризики через автоматичний вибір стовпців для об'єднання.

Однією з основних проблем NATURAL JOIN є його неявність: ви не контролюєте, за якими саме стовпцями відбувається об'єднання. Це може призвести

до непередбачуваних результатів, особливо якщо таблиці мають багато стовпців з однаковими іменами, але потребують об'єднання по специфічних стовпцях.

Тому багато розробників уникають використання NATURAL JOIN і вибирають явне визначення стовпців для об'єднання за допомогою звичайних JOIN або INNER JOIN, де вони вказують стовпці, за якими треба проводити об'єднання, що робить код більш зрозумілим і передбачуваним.

Запит:

```
SELECT * FROM `TableA`
NATURAL JOIN `TableB`
```

В цьому випадку СУБД вибирає для об'єднання таблиць стовпчики id і name, оскільки вони присутні в обидвох таблицях та здійснює перетворення початкового запиту в запит наступного виду:

```
SELECT * FROM `TableA`
INNER JOIN `TableB`
USING (`id`, `name`)
```

Оскільки ми немаємо записів з однаковим id і name одночасно в обох таблицях, то запит поверне пустий результат. Однак якщо зробити ліву таблицю керівною і дещо змінити запит:

```
SELECT `TableA`.*, `TableB`.* FROM `TableA`
NATURAL LEFT JOIN `TableB`
```

Такий запит призведе до:

```
SELECT `TableA`.*, `TableB`.* FROM `TableA`
LEFT JOIN `TableB`
USING (`id`, `name`)
```

NATURAL JOIN, хоч і видається зручним у певних випадках, не завжди є найкращим вибором через свою непередбачуваність. Основні причини, чому розробники уникають використання NATURAL JOIN:

1. **Непередбачуваність об'єднання:** NATURAL JOIN автоматично вибирає стовпці для об'єднання, що може призвести до несподіваних результатів, особливо коли структура бази даних змінюється або містить багато спільних стовпців.
2. **Читабельність коду:** Код з NATURAL JOIN може бути менш зрозумілим, оскільки неявно вказує, як саме відбувається об'єднання таблиць. Явне вказання стовпців для об'єднання за допомогою INNER JOIN або звичайного JOIN забезпечує більшу ясність та передбачуваність.
3. **Гнучкість:** Явне вказання стовпців для об'єднання дає більшу гнучкість у визначенні, які саме стовпці повинні бути ключовими для об'єднання.

Тому багато розробників уникають використання NATURAL JOIN і віддають перевагу явному вказанню стовпців для об'єднання через INNER JOIN або звичайний JOIN. Це зробить код більш прозорим, передбачуваним і легким для розуміння для інших розробників, які будуть працювати з вашим кодом.

STRAIGHT JOIN

STRAIGHT JOIN виконує ті самі функції, що й простий INNER JOIN, проте ліва таблиця читається раніше правої.

Запит:

```
SELECT * FROM `TableA`
STRAIGHT JOIN `TableB` USING(`name`)
```

Запит:

```
SELECT * FROM `TableB`
STRAIGHT JOIN `TableA` USING(`name`)
```

Якщо при об'єднанні таблиць не вказати умову об'єднання (наприклад, через ON або USING), то СУБД виконає Декартовий добуток (CROSS JOIN). Це означає, що кожен рядок з першої таблиці буде об'єднаний з кожним рядком з другої таблиці, утворюючи всі можливі комбінації, і результатом буде набір даних, кількість рядків

якого дорівнює добутку кількості рядків у кожній таблиці 4x4 = 16 рядків.

Це важливо враховувати при написанні запитів, оскільки Декартовий добуток може створювати величезну кількість рядків у випадках, коли таблиці мають багато записів. Якщо така кількість комбінацій не потрібна, необхідно явно вказати умови об'єднання для точного об'єднання таблиць, щоб уникнути непотрібних результатів Декартового добутку.

Запит:

```
SELECT * FROM `TableA`
JOIN `TableB`
```

3.3. Висновки до розділу

Системи управління реляційними базами даних (СУРБД) стали невід'ємною частиною бізнес-інфраструктури багатьох компаній. Реляційна модель даних дозволяє ефективно зберігати та управляти інформацією через використання таблиць зі зв'язками між ними.

Об'єднання реляційних баз даних з клієнт-серверною архітектурою має численні переваги. Клієнт-серверна система дозволяє розподілене зберігання даних на серверах, забезпечуючи доступ до цих даних через мережу для користувачів. Це не лише забезпечує швидкий доступ до інформації, а й робить систему більш гнучкою, безпечною та масштабованою.

Така архітектура дозволяє багатьом користувачам одночасно працювати з даними, керувати ними та забезпечувати конкурентні переваги підприємству через ефективне використання та обробку інформації.

Власне завдяки системі управління реляційними базами данних AiDocs є інтелектуальним інструментом, що надає користувачам зручний функціонал для генерації нових шаблонів на основі існуючих. Це дозволяє прискорити процес створення документів, оскільки користувач може використовувати вже існуючі шаблони як основу і адаптувати їх до своїх потреб.

РОЗДІЛ 4. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1. Розроблення інтелектуальної ситеми створення шаблонів підприємця.

AiDocs - це інноваційна система для створення та управління документами, розроблена на базі фреймворку React. Цей інтелектуальний інструмент надає користувачам можливість легко генерувати нові шаблони на основі існуючих, що робить процес створення документів швидким та ефективним.

Завдяки великій базі шаблонів AiDocs користувачі можуть вибирати ідеальний шаблон для своїх потреб та легко його налаштовувати. Система надає можливість роздрукувати створені документи або завантажити їх у форматі PDF. Однак ці функції доступні лише для зареєстрованих користувачів, що додає додатковий рівень безпеки та конфіденційності.

Для незареєстрованих користувачів AiDocs надає можливість переглядати захищені водяними знаками зображення шаблонів документів. Це дозволяє користувачам оцінити функціональність та різноманіття програми перед реєстрацією.

Використання React[див. рис. 1]. у розробці AiDocs гарантує високу реактивність та зручний інтерфейс для користувачів. Компонентний підхід дозволяє легко взаємодіяти з різними функціями системи та ефективно користуватися її можливостями.

Узагальнюючи, Ai Docs є інноваційним інструментом, що допомагає легко та швидко створювати різноманітну документацію, забезпечуючи високий рівень зручності та функціональності для своїх користувачів.AiDocs - це не просто інструмент для створення документів, але і потужна платформа з усіма необхідними інструментами для продуктивної роботи з документацією.

Рис.1 React.

Заснована на принципах компонентного підходу, система AiDocs розбиває веб-інтерфейс на компоненти, які можуть бути легко перевикористовані та модифіковані. Це сприяє простоті розробки та обслуговування, а також дозволяє розробникам ефективно співпрацювати над проектом.

Однією з ключових переваг використання React в AiDocs є використання віртуального DOM, що призводить до швидкого та ефективного відображення змін в інтерфейсі. Цей підхід допомагає підтримувати високу продуктивність додатку та зменшує навантаження на браузер користувача.

AiDocs використовує принцип одностороннього потоку даних для ефективного відстеження та керування станом компонентів. Це забезпечує послідовність та передбачуваність роботи, а також полегшує виявлення та виправлення помилок.

Завдяки розширюваності React та додаткових бібліотек, таких як Redux та React Router, AiDocs може легко адаптуватися та розширюватися для надання більших можливостей та функціональності. Активне співтовариство розробників робить цей інструмент ще потужнішим, забезпечуючи велику кількість ресурсів, підтримки та документації.

AiDocs визначається як інноваційний та прогресивний інструмент для тих, хто шукає ефективний спосіб створення та управління бізнес-документами.

Перш за все, React надає нам можливість створювати компоненти, які можна повторно використовувати і легко управляти. Це дозволяє нам побудувати складну структуру веб-сайту з простих блоків, що спрощує розробку та підтримку коду.

React також надає нам можливість управляти станом додатка за допомогою компонентів і хуків, таких як useState і useEffect. Це дозволяє нам реагувати на події та зміни даних у реальному часі, оновлюючи відображення відповідно до нових значень стану. Такий підхід дозволяє нам створювати інтерактивні та реактивні елементи нашої платформи.

Крім того, за допомогою React ми можемо легко інтегрувати інші бібліотеки та розширення, що дозволяє нам забезпечити додатковий функціонал та розширити можливості нашої платформи. Ми можемо використовувати багато готових компонентів, стилізаційних бібліотек та інших інструментів, що полегшують розробку та покращують вигляд нашого веб-сайту.

Усі ці особливості React доповнюють нашу програмну частину інтелектуальної ситеми "AI DOCS" унікальними та потужними можливостями, що дозволяють нам створити сучасну, швидку та зручну платформу для підприємців.

Створення ситеми включає кілька кроків, які допомагають забезпечити успішну розробку та запуск веб-проекту. Нижче описані основні етапи процесу створення сайту системи[див. рис. 2].

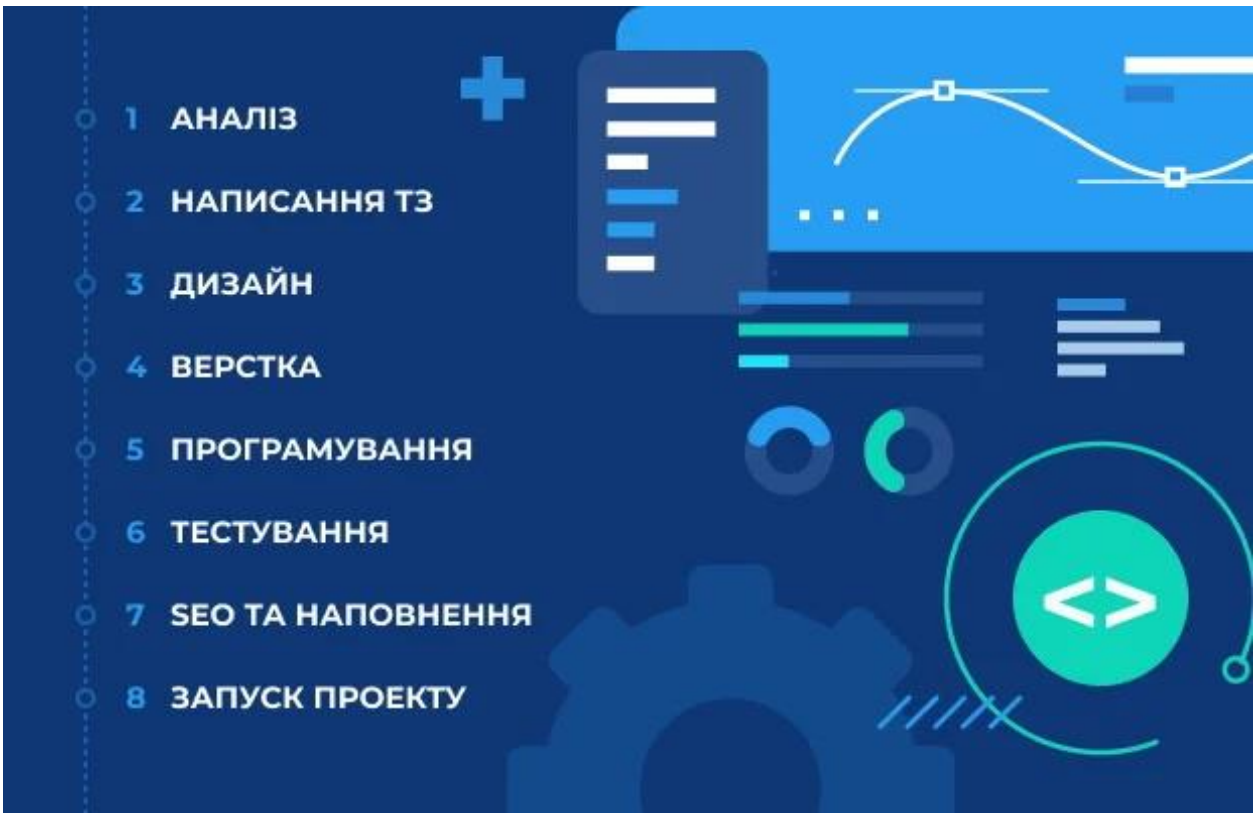


Рис.2 Етапи розробки

Аналіз: Спочатку чітко аналізуємо та визначаємо мету веб-сайту. Потрібно зрозуміти, яку функціональність потрібно надати користувачам, який тип веб-сайту створити і які результати потрібно досягти.

Написання ТЗ: Наступним кроком є планування структури веб-сайту і проектування його вигляду та написання технічного завдання.

Дизайн: Можна створити візуальні макети, скетчі або використовувати дизайн-систему для створення концепції та розташування елементів на сторінках сайту.

Верстка: За допомогою HTML, CSS створюється фронтенд веб-сайту. HTML використовується для створення структури сторінок, CSS - для стилізації та оформлення.

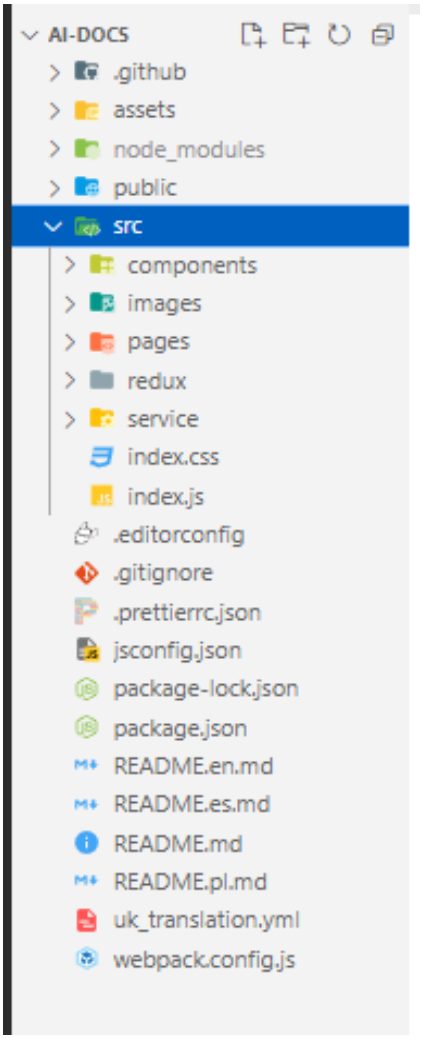
Програмування: За допомогою React наповнюємо вебсайт функціональністю та логікою.

Тестування: Після завершення розробки тестуємо веб-сайт, щоб переконатися, що всі функції працюють належним чином і немає помилок. Тестування можна проводити вручну або автоматизовано.

Розгортання: Після успішного тестування веб-сайт готовий до розгортання. Це означає, що ви розміщуєте ваш веб-сайт на сервері або хостинг-платформі, щоб він став доступним для користувачів через Інтернет.

Seo та наповнення: Наповнюємо базу даних сайту

Запуск: Після запуску веб-сайту важливо забезпечити його підтримку та здійснювати необхідні поновлення. Це може включати виправлення помилок, вдосконалення функціональності, оновлення залежностей тощо.



При розробці веб-сайту з використанням React, проект має наступну файлову структуру[див. рис 3]:

src/index.js: Цей файл є основним файлом у React-програмі та відповідає за відображення вашого React-компоненту у вигляді HTML-елементу на веб-сторінці. Зазвичай в ньому використовується метод ReactDOM.render() для включення головного компонента.

src/App.js: Цей файл є контейнером для головного компоненту вашого додатку. Ви можете розширювати його функціональність або використовувати як основний контейнер для інших компонентів на вашому веб-сайті.

src/components: У цьому каталозі зазвичай знаходяться всі компоненти вашого веб-сайту. Кожен компонент може бути розміщений у власному файлі, який включає в себе код компонента та, можливо, окремий файл із стилями.

Рис.3 Структура проекту

src/services: Якщо у вас передбачена взаємодія з сервером або зовнішніми API, рекомендується створити каталог "services". У цьому каталозі можна

розмістити файли, що містять логіку взаємодії з вказаними сервісами або API. Це допомагає структурувати ваш проект та розділити логіку взаємодії з сервером від компонентів і їхньої логіки.

src/store: Якщо ви використовуєте стейт-менеджер, наприклад, Redux, для ефективного керування станом вашого додатку, то можна створити каталог з назвою "store". У цьому каталозі розмістіть файли, пов'язані зі станом (reducers), діями (actions) та, можливо, іншими елементами, пов'язаними із стейт-менеджментом вашого додатку. Це допомагає утримувати структуру проекту чистою та організованою.

Під час розробки веб-сайту на React підключені наступні бібліотеки та інструменти:

ReactDOM - це пакет, що використовується для рендерингу React-компонентів у реальний DOM. Цей інструмент взаємодіє з браузером та дозволяє оновлювати та відображати компоненти на сторінці. Завдяки ReactDOM процес рендерингу стає простішим та ефективнішим, надаючи можливість React-компонентам взаємодіяти з реальним DOM-деревом.

```
import Home from 'pages/Home/Home';
import { Route, Routes } from 'react-router-dom';
import Layout from './Layout/Layout';
import About from 'pages/About/About';
import Contacts from 'pages/Contacts/Contacts';
import Templates from 'pages/Templates/Templates';

import TemplateEditor from 'pages/TemplateEditor/TemplateEditor';

export const App = () => {
  return (
    <Routes>
      <Route path="/" element={<Layout />}>
        <Route index element={<Home />} />
        <Route path="template-edit" element={<TemplateEditor />} />
        <Route path="template-edit/:id" element={<TemplateEditor />} />
        <Route path="templates" element={<Templates />} />
        <Route path="about" element={<About />} />
        <Route path="contacts" element={<Contacts />} />
      </Route>
    </Routes>
  ); ← #11-22 return
}; ← #10-23 export const App = () =>
```

Рис.4 React Router

React Router - це бібліотека, призначена для маршрутизації в React-додатках. Вона дозволяє визначати маршрути та пов'язані з ними компоненти, які мають бути відображені для кожного маршруту. За допомогою React Router можна легко створювати односторінкові додатки з багатьма сторінками та налаштовувати навігацію між ними. Бібліотека також підтримує параметри маршруту, вкладені маршрути та зберігає історію переходів. [див. рис 4].

Redux: Redux є високопопулярною бібліотекою для керування станом додатків, розроблених на React. Вона ґрунтується на архітектурі Flux та допомагає організувати та контролювати глобальний стан додатка. Redux використовує одне центральне сховище (store), в якому зберігається весь стан додатка. Компоненти можуть звертатись до цього сховища для отримання стану та відправляти дії (actions), що змінюють стан. Redux спрощує керування станом, забезпечуючи передбачувану поведінку та полегшуючи відлагодження. [див. рис 5].

```
} from 'redux-persist'; ← #2-11 import
import { authSlice } from './auth/authSlice';
import { templatesSlice } from './templates/templatesSlice';

import storage from 'redux-persist/lib/storage'; // defaults to localStorage for web

const middleware = [
  ...getDefaultMiddleware({
    serializableCheck: {
      ignoredActions: [FLUSH, REHYDRATE, PAUSE, PERSIST, PURGE, REGISTER],
    },
  }), ← #18-22 ...getDefaultMiddleware
]; ← #17-23 const middleware =

const authPersistConfig = {
  key: 'auth',
  storage,
};

const persistedAuthReducer = persistReducer(
  authPersistConfig,
  authSlice.reducer
);

const templatesPersistConfig = {
  key: 'templates',
  storage,
  whitelist: ['templates', 'favTemplates'],
}; ← #35-39 const templatesPersistConfig =

const persistedItemsReducer = persistReducer(
  templatesPersistConfig,
  templatesSlice.reducer
);

export const store = configureStore({
  reducer: {
    auth: persistedAuthReducer,
    templates: persistedItemsReducer,
  },
  middleware,
  devTools: process.env.NODE_ENV === 'development',
}); ← #46-53 export const store = configureStore

export const persistor = persistStore(store);    "persistor": Unknown word.
```

Рис.5 Redux

Файл package.json є ключовим елементом проектів, розроблених на основі

Node.js, включаючи проекти React. Цей файл містить метадані про проект та його залежності, а також конфігураційні налаштування[див. рис 6].

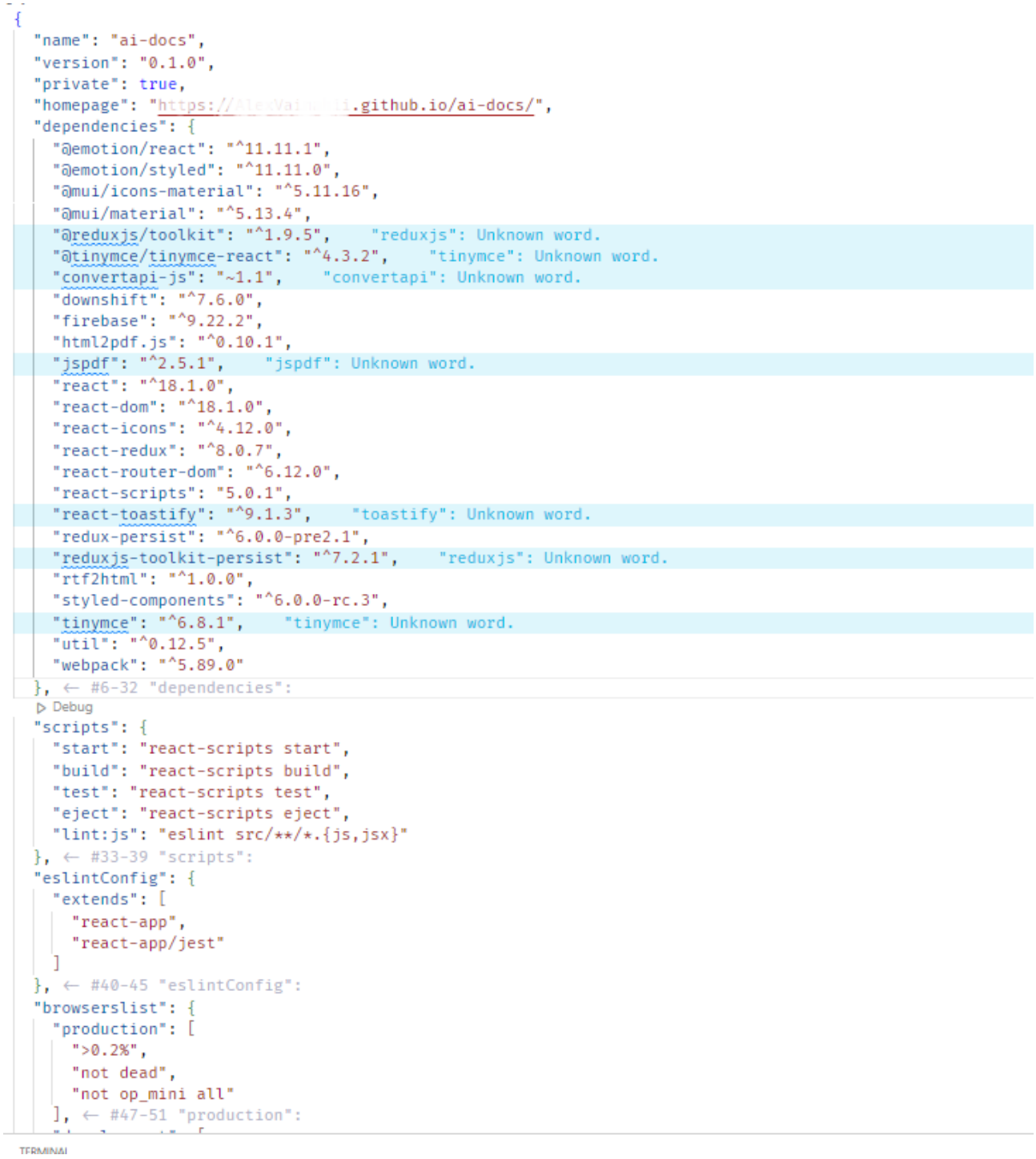


Рис.6 Package.json

Styled Components: Styled Components є потужною бібліотекою, яка дозволяє стилізувати React-компоненти за допомогою підходу CSS-in-JS. Замість традиційного використання зовнішніх CSS-файлів, Styled Components дозволяє

вбудовувати стилі прямо в код компонента. Ця бібліотека пропонує зручний синтаксис для опису стилів, при цьому автоматично генерує унікальні CSS-класи для кожного компонента, що підвищує ізольованість стилів та уникає конфліктів.

Однією з ключових переваг Styled Components є його підтримка динамічних стилів. Це означає, що ви можете змінювати стилі компонентів в залежності від умов або стану додатка, дозволяючи створювати більш гнучкий та динамічний інтерфейс.

Крім того, Styled Components легко інтегрується з темами, дозволяючи легко змінювати вигляд вашого додатка за допомогою глобально визначених стилів для різних елементів.

Бібліотека також дозволяє легко визначати анімації, що робить її потужним інструментом для створення привабливих та динамічних інтерфейсів в React-додатках. [див. рис 7].

```
import styled from 'styled-components';
import SideImage from '../../images/fon.jpg';
import { NavLink } from 'react-router-dom';

const Container = styled.div`
  display: flex;
  flex-wrap: no-wrap;
`;

const Sidebar = styled.div`
  width: 300px;
  background-image: url(${SideImage});
  background-repeat: no-repeat;
  background-size: cover;
  background-position: 0% 50%;

  height: 98vh;
  display: flex;
  flex-direction: column;

  box-shadow: 0px 0px 25px 0px rgba(0, 0, 0, 0.7);
`;

const Header = styled.div`
  padding: 15px;
  display: flex;
  justify-content: center;
`;

const ResponsiveLayout = styled.div`
  @media (max-width: 468px) {
    ${Container} {
      flex-direction: column;
    }
    ${Sidebar} {
      width: 100%;
    }
  }
`;
export const Navigation = styled.nav`
  margin-top: 20px;
  display: flex;
  flex-direction: column;
  width: 80%;
  margin-left: 25px;
  z-index: 5;
`;
```

Рис.7 Styled Components

Firebase:Firebase є потужною платформою для розробки веб- та мобільних додатків, яка надає широкий спектр інструментів та сервісів. Основні можливості Firebase включають:

Збереження та синхронізація даних в реальному часі: Firebase надає базу даних в реальному часі, яка автоматично синхронізує дані між всіма підключеними клієнтами. Це дозволяє легко створювати додатки, які реагують на зміни даних миттєво.

Автентифікація користувачів: Firebase має вбудований сервіс аутентифікації, який спрощує ідентифікацію та авторизацію користувачів. Ви можете використовувати різні методи автентифікації, такі як електронна пошта, соціальні мережі та інші.

Повідомлення в режимі реального часу: Firebase Cloud Messaging дозволяє вам надсилати повідомлення на мобільні пристрої та веб-додатки в режимі реального часу. Це корисно для реалізації сповіщень та взаємодії з користувачами.

Хостинг та CDN: Firebase дозволяє легко розгортати ваш веб-сайт або додаток за допомогою вбудованого хостингу та глобальної мережі CDN, що поліпшує швидкодію завантаження ресурсів.

Функції від сервера: Firebase дозволяє написання та розгортання власних функцій від сервера, що виконуються в хмарі. Це може бути корисно для виконання обчислень на стороні сервера або інтеграції з іншими службами.

Інструменти для аналітики та моніторингу: Firebase пропонує інструменти для відстеження та аналізу використання додатків, що допомагає вам зрозуміти поведінку користувачів.

Інтеграція Firebase з React-додатками виконується за допомогою спеціального SDK для JavaScript, який надає зручний інтерфейс для взаємодії з різними сервісами Firebase у вашому додатку. Використання Firebase полегшує багато аспектів розробки веб-сайтів, таких як робота з базою даних, аутентифікація користувачів та взаємодія з сервером, дозволяючи розробникам зосередитися на функціональності свого додатку. [див. рис 8,9,10 ,11,12,13,14,].

```
import {
  collection,
  addDoc,
  deleteDoc,
  getDocs,
  doc,
  getFirestore,    "Firestore": Unknown word.
  updateDoc,
} from 'firebase/firestore';    "firestore": Unknown word. ← #5-13 import
```

```
import { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';

const firebaseConfig = {
  apiKey: 'AIzaSyAXxd5BLFD3cjyRh4yRnr-CKRAsYniXUek',    "BLFD": Unknown word.
  authDomain: 'ai-docs-1807d.firebaseapp.com',    "firebaseapp": Unknown word.
  projectId: 'ai-docs-1807d',
  storageBucket: 'ai-docs-1807d.appspot.com',    "appspot": Unknown word.
  messagingSenderId: '722774303553',
  appId: '1:722774303553:web:8f91361d17034c050bf7c0',
}; ← #15-22 const firebaseConfig =

// Initialize Firebase
const app = initializeApp(firebaseConfig);

// Initialize Firebase Authentication and get a reference to the service
const auth1 = getAuth(app);

// Створюємо провайдера Google для авторизації через Google    "Створюємо": Unknown word.
const googleProvider = new GoogleAuthProvider();
```

Рис.8 Firebase ініціалізація в проекті

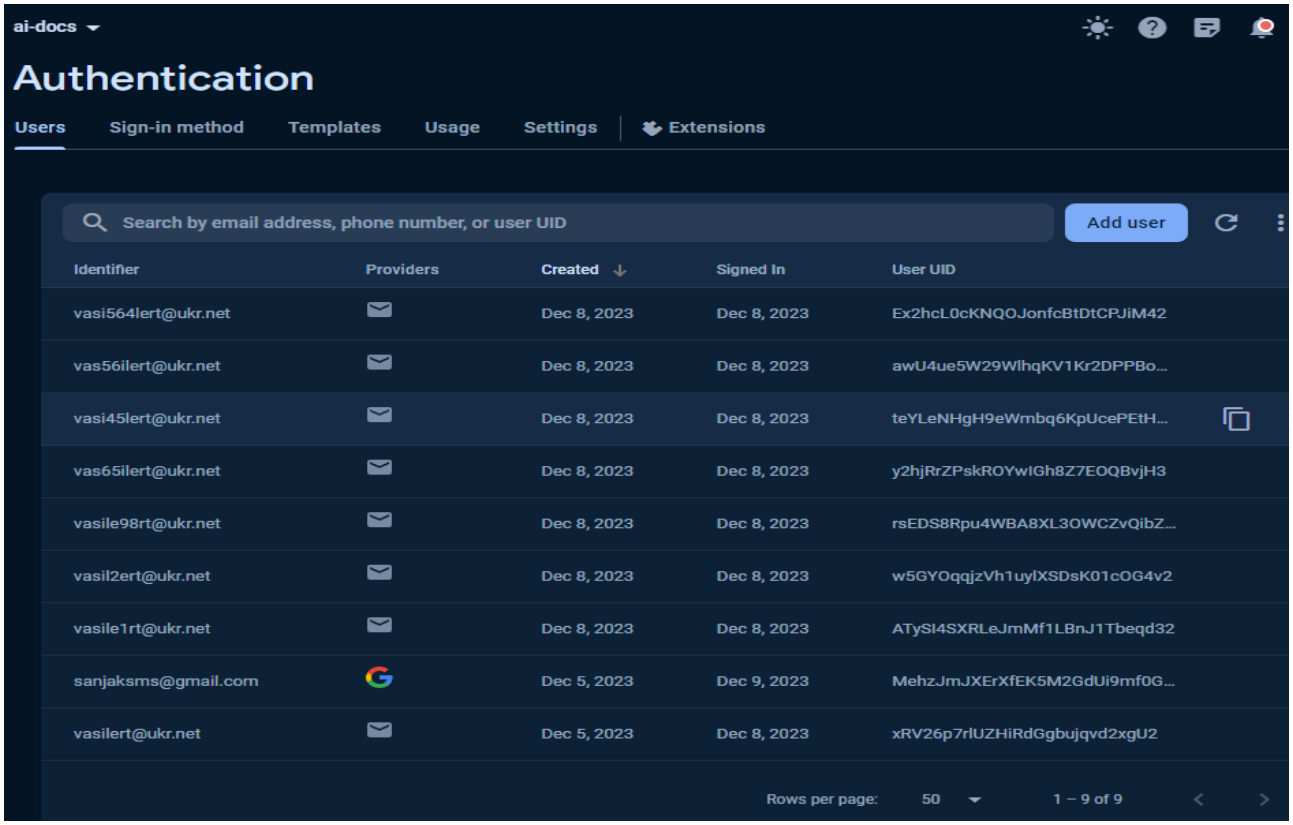


Рис.9 Firebase Аутентифікація користувачів

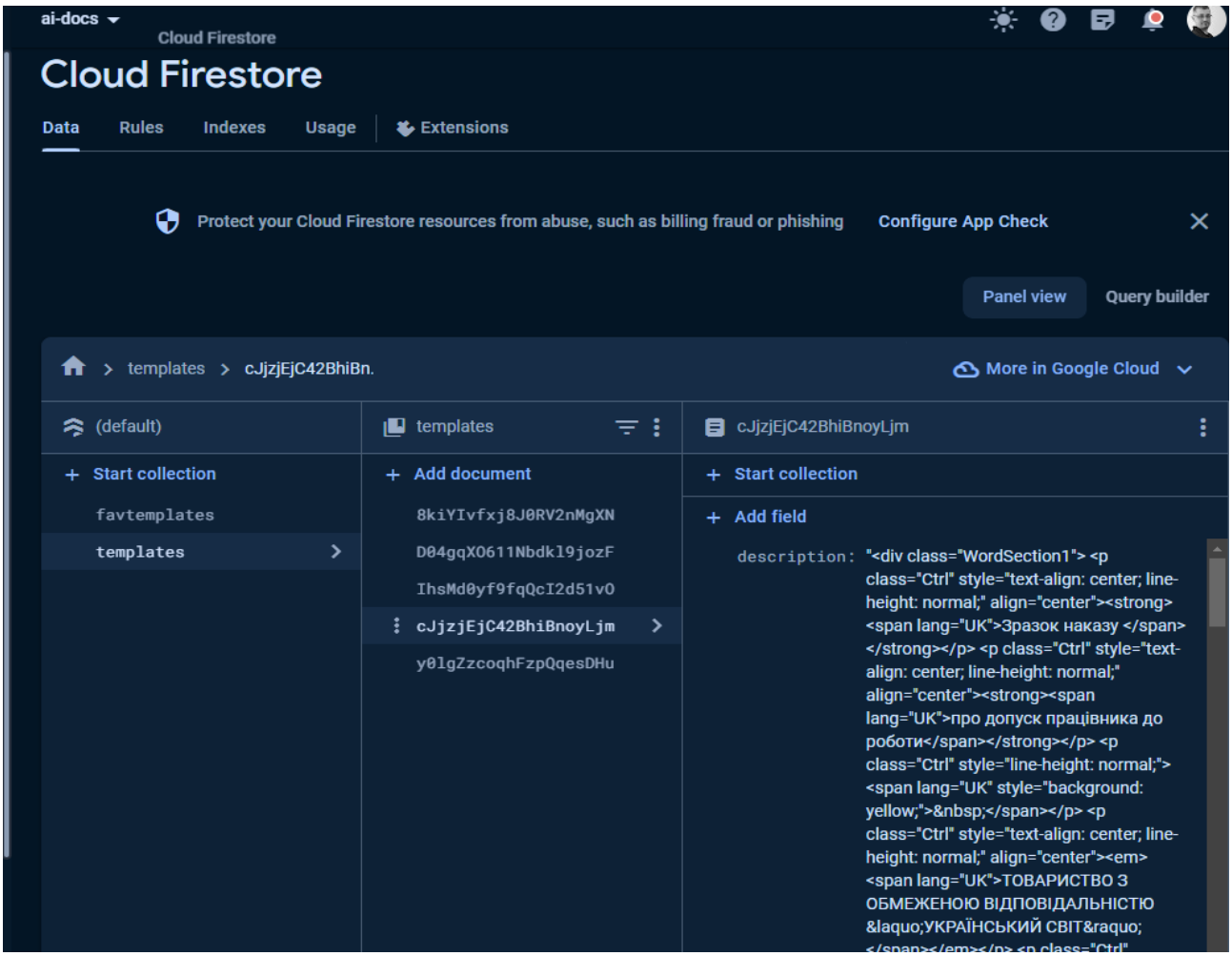


Рис.10 Firebase Збереження даних на сервері

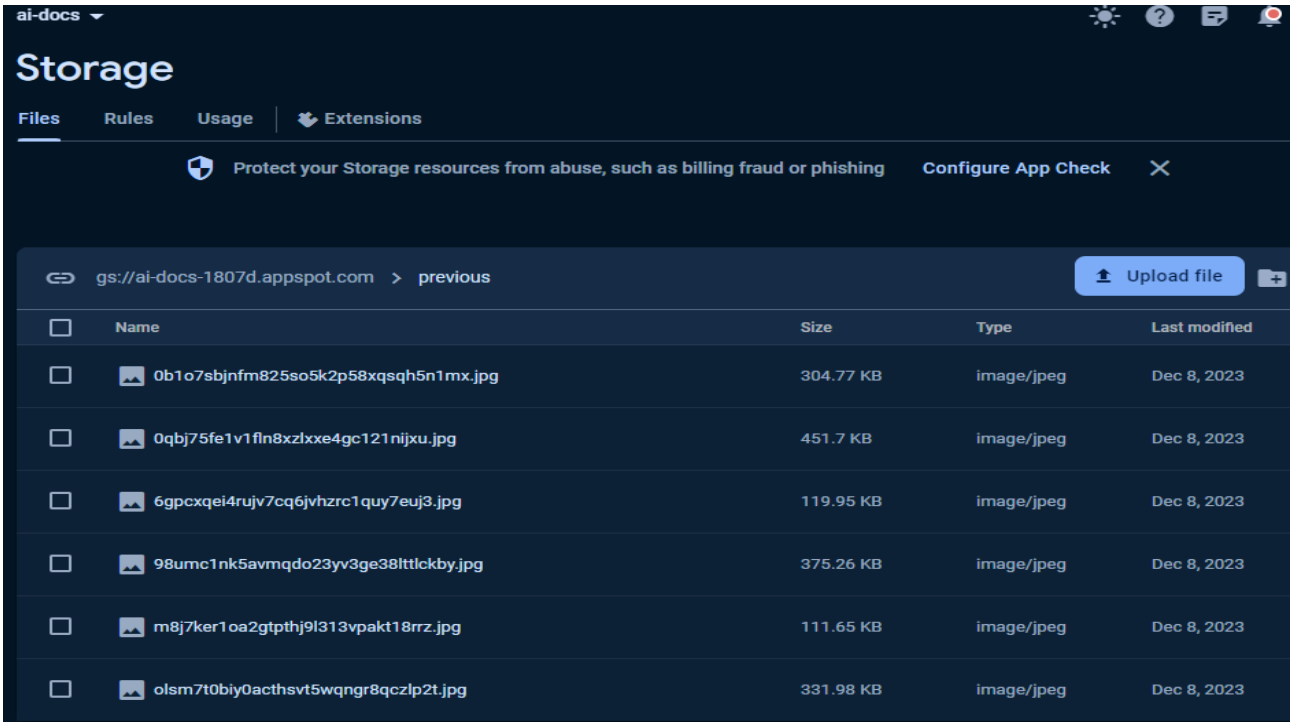


Рис.11 Firebase Збереження файлів на сервері

```
export const register = createAsyncThunk(
  'auth/register',
  async ({ email, password }, { rejectWithValue }) => {
    try {
      const userCredential = await createUserWithEmailAndPassword(
        auth1,
        email,
        password
      ); ← #37-41 const userCredential = await createUserWithEmailAndPassword
      const user = userCredential.user;
      return user;
    } catch (e) {
      toast.error(`User with email ${email} is already registered!`);

      return rejectWithValue(e.message);
    } ← #44-48 catch (e)
  } ← #35-49 async ({ email, password }, { rejectWithValue }) =>
); ← #33-50 export const register = createAsyncThunk
export const login = createAsyncThunk(
  'auth/login',
  async ({ email, password }, { rejectWithValue }) => {
    try {
      const userCredential = await signInWithEmailAndPassword(
        auth1,
        email,
        password
      ); ← #55-59 const userCredential = await signInWithEmailAndPassword
      const user = userCredential.user;
      return user;
    } catch (e) {
      toast.error(`User ${email} is login error.`);

      return rejectWithValue(e.message);
    } ← #62-66 catch (e)
  } ← #53-67 async ({ email, password }, { rejectWithValue }) =>
); ← #51-68 export const login = createAsyncThunk
export const loginWithGoogle = createAsyncThunk(
  'auth/loginWithGoogle',
  async (_, { rejectWithValue }) => {
    try {
      const userCredential = await signInWithPopup(auth1, googleProvider);
      const { displayName, email } = userCredential.user;

      return { displayName, email };
    } catch (e) {
      console.log('e :>> ', e);
      toast.error(`User  is login error.`);

      return rejectWithValue(e.message);
    } ← #77-82 catch (e)
  } ← #71-83 async (_, { rejectWithValue }) =>
); ← #69-84 export const loginWithGoogle = createAsyncThunk
export const logout = createAsyncThunk(
  'auth/logout',
  async (_, { rejectWithValue }) => {
    try {
      await signOut(auth1);
      return;
```

Рис.12 Firebase Операції аутентифікації

```
export const storage = getStorage(app);

export const setImage = async (dirname, uri, uid) => {
  const data = await fetch(uri);
  const photo = await data.blob();
  const storagePhotoRef = ref(storage, `${dirname}/${uid}.jpg`);
  try {
    const snapshot = await uploadBytes(storagePhotoRef, photo);
    const url = await getDownloadURL(snapshot.ref);
    return url;
  } catch (error) {
    console.log('error :>> ', error);
  }
}; ← #27-38 export const setImage = async (dirname, uri, uid) =>
```

Рис.13 Firebase Операції збереження файлів

```
export const fetchTemplates = createAsyncThunk(
  'template/fetchAll',
  async (_, thunkAPI) => {
    try {
      const templateCollection = collection(db, 'templates');
      const querySnapshot = await getDocs(templateCollection);

      const templates = [];
      querySnapshot.forEach(template => {
        templates.push({ id: template.id, ...template.data() });
      });
      const favtemplateCollection = collection(db, 'favtemplates');    "favtemplate": Unknown word.
      const querySnapshot1 = await getDocs(favtemplateCollection);    "favtemplate": Unknown word.

      const favtemplates = [];    "favtemplates": Unknown word.
      querySnapshot1.forEach(template => {
        favtemplates.push({ id: template.id, ...template.data() });    "favtemplates": Unknown word.
      });

      return { templates, favtemplates };    "favtemplates": Unknown word.
    } catch (e) {
      return thunkAPI.rejectWithValue(e.message);
    }
  } ← #41-62 async (_, thunkAPI) =>
); ← #39-63 export const fetchTemplates = createAsyncThunk

export const addTemplate = createAsyncThunk(
  'templates/addTemplate',
  async ({ userEmail, name, description, templateImageUrl }, thunkAPI) => {
    try {
      const templatesCollection = collection(db, 'templates');

      const docRef = await addDoc(templatesCollection, {
        name,
        description,
        userEmail,
        templateImageUrl,
      }); ← #71-76 const docRef = await addDoc
      toast.success(`Templete added`);    "Templete": Unknown word.
      return { id: docRef.id, userEmail, name, description };
    } catch (e) {
      return thunkAPI.rejectWithValue(e.message);
    }
  } ← #67-82 async ({ userEmail, name, description, templateImageUrl }, th...
); ← #65-83 export const addTemplate = createAsyncThunk
```

Рис.14 Firebase Операції додавання даних до бази

Ці бібліотеки та інструменти значно полегшують процес створення веб-сайтів на React та надають розробникам широкий набір корисних можливостей. Використання цих інструментів спрощує написання чистого, ефективного та легко керованого коду, що призводить до швидкішої розробки та забезпечує зручну підтримку веб-додатків.

Однією з важливих задач для мене була розробка унікального компонента для автентифікації на сайті. Цей компонент побудований на базі бібліотек React та Redux і включає набір функцій для управління процесом автентифікації користувачів.

Основні можливості цього компонента включають:

- Відображення форми для введення електронної пошти та пароля.
- Можливість реєстрації нового користувача.
- Можливість входу в систему за допомогою введених даних.
- Підтримка автентифікації через обліковий запис Google.
- Відображення повідомлень про помилки або успішні операції.
- Відображення інформації про користувача та кнопки для виходу з системи.
- Можливість переключення між формою автентифікації та іншими режимами.

Цей компонент є гнучким та може бути використаний на будь-якій сторінці сайту, де потрібна функціональність автентифікації користувачів. Він пропонує зручний інтерфейс та надає можливість взаємодіяти з різними типами користувачів, дозволяючи їм реєструватися, увійти в систему або вийти, використовуючи різні способи автентифікації.

Цей компонент є унікальним завдяки своїй реалізації на основі сучасних технологій та використанню найкращих практик веб-розробки. Він може служити ключовим елементом вашого сайту, додаючи до нього потужну функціональність автентифікації та забезпечуючи безпеку та захист конфіденційної інформації користувачів. [див. рис 15,16].

```
const AuthForm = () => {
  const navigate = useNavigate();
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const showAuthForm = useSelector(selectShowAuthForm);
  const loading = useSelector(selectLoading);
  const error = useSelector(selectError);
  const successMessage = useSelector(state => state.auth.successMessage);
  const dispatch = useDispatch();

  const handleToggleAuthForm = () => {
    dispatch(setShowAuthForm(!showAuthForm));
  };

  const handleRegister = async e => {
    e.preventDefault();

    dispatch(register({ email, password }));
  }; ← #48-52 const handleRegister = async e =>

  const handleLogin = async e => {
    e.preventDefault();
    dispatch(login({ email, password }));
  };

  const handleGoogleLogin = async e => {
    e.preventDefault();
    dispatch(loginWithGoogle());
  };

  const handleLogout = async () => {
    setEmail('');
    setPassword('');
    dispatch(setShowAuthForm(false));
    navigate('/');
    dispatch(logout());
  }; ← #64-70 const handleLogout = async () =>

  const user = useSelector(selectUser);
```

Рис.15 Функції роботи форми аутентифікації

```
return (
  <AuthFormContainer>
    {loading ? (
      <p>Loading ... </p>
    ) : (
      ◇
        {error && <AuthFormErrorMessage>{error}</AuthFormErrorMessage>}
        {successMessage && (
          <AuthFormSuccessMessage>{successMessage}</AuthFormSuccessMessage>
        )}
        {!user && !showAuthForm ? (
          <AuthFormButton
            onClick={handleToggleAuthForm}
            title="Увійти в систему"    "Увійти": Unknown word.
          >
            <AccountCircleIcon />
          </AuthFormButton>
        ) : (
          ◇
            {!user ? (
              <form>
                <AuthFormInput
                  type="email"
                  value={email}
                  onChange={e ⇒ setEmail(e.target.value)}
                  placeholder="Email"
                />
                <AuthFormInput
                  type="password"
                  value={password}
                  onChange={e ⇒ setPassword(e.target.value)}
                  placeholder="Password"
                />
                <AuthFormButtonContainer>
                  <AuthFormButton onClick={handleRegister} title="Реєстрація">    "Реєстрація": Unknown word.
                    <AppRegistrationIcon />
                  </AuthFormButton>
                  <AuthFormButton onClick={handleLogin} title="Увійти">    "Увійти": Unknown word.
                    <LoginIcon />
                  </AuthFormButton>
                  <AuthFormButton
                    onClick={handleGoogleLogin}
                    title="Увіти за допомого Google-акаунта"    "Увіти": Unknown word.
                  >
                    <GoogleIcon />
                  </AuthFormButton>
                  <AuthFormButton
                    onClick={handleToggleAuthForm}
                    title="Відмінити"    "Відмінити": Unknown word.
                  >
                    <CancelIcon />
                  </AuthFormButton>
                </AuthFormButtonContainer>
              </form>
```

Рис.16 Код відображення форми аутентифікації

Також для автентифікації та збереження даних була використана бібліотека Redux-Toolkit, яка є популярним інструментом для керування станом додатків на основі Redux в середовищі React.

Redux-Toolkit надає простий та зручний спосіб організації стану додатку, забезпечуючи централізоване збереження даних та простий доступ до них з будь-якого місця в додатку. Вона поєднує у собі декілька інших бібліотек Redux, таких як Redux-Thunk та Redux-Saga, що дозволяє працювати з асинхронними діями та ефектами.

Основні переваги використання Redux-Toolkit включають:

Спрощений синтаксис: Redux-Toolkit надає спрощений синтаксис для оголошення дій, редюсерів та створення централізованого стору. Це дозволяє зменшити кількість коду, необхідного для роботи з Redux.

Вбудована підтримка асинхронності: Redux-Toolkit має вбудовану підтримку для виконання асинхронних операцій за допомогою Redux-Thunk або Redux-Saga. Це дозволяє легко виконувати запити до Firebase API або інших зовнішніх сервісів.

Інструменти розробника: Redux-Toolkit надає набір корисних інструментів для розробника, таких як Redux DevTools, які спрощують налагодження та відлагодження додатку, а також надають інформацію про зміни стану додатку у реальному часі.

Модульність: Redux-Toolkit дозволяє організувати код додатку в модулі, що полегшує його розширення та підтримку. Кожен модуль може містити свій власний набір дій, редюсерів та селекторів, що полегшує розподіл роботи між розробниками та підтримку масштабованих додатків.

Завдяки використанню Redux-Toolkit, автентифікаційний компонент може ефективно керувати станом автентифікації, зберігати дані користувачів та взаємодіяти з Firebase для автентифікації та збереження даних. Це дозволяє зробити додаток більш масштабованим, керованим та ефективним з точки зору керування станом [див. рис 17-20].

```
import { configureStore, getDefaultMiddleware } from '@reduxjs/toolkit';    "reduxjs": Unkno
import {
  persistStore,
  persistReducer,
  FLUSH,
  REHYDRATE,
  PAUSE,
  PERSIST,
  PURGE,
  REGISTER,
} from 'redux-persist'; ← #2-11 import
import { authSlice } from './auth/authSlice';
import { templatesSlice } from './templates/templatesSlice';

import storage from 'redux-persist/lib/storage'; // defaults to localStorage for web

const middleware = [
  ...getDefaultMiddleware({
    serializableCheck: {
      ignoredActions: [FLUSH, REHYDRATE, PAUSE, PERSIST, PURGE, REGISTER],
    },
  }), ← #18-22 ...getDefaultMiddleware
]; ← #17-23 const middleware =

const authPersistConfig = {
  key: 'auth',
  storage,
};

const persistedAuthReducer = persistReducer(
  authPersistConfig,
  authSlice.reducer
);

const templatesPersistConfig = {
  key: 'templates',
  storage,
  whitelist: ['templates', 'favTemplates'],
}; ← #35-39 const templatesPersistConfig =

const persistedItemsReducer = persistReducer(
  templatesPersistConfig,
  templatesSlice.reducer
);

export const store = configureStore({
  reducer: {
    auth: persistedAuthReducer,
    templates: persistedItemsReducer,
  },
  middleware,
  devTools: process.env.NODE_ENV === 'development',
}); ← #46-53 export const store = configureStore

export const persistor = persistStore(store);    "persistor": Unknown word.
```

Рис.17 Головний «Стор» сайту

```
import { createSlice } from '@reduxjs/toolkit';    "reduxjs": Unknown word.
import {
  login,
  loginWithGoogle,
  logout,
  register,
  userRefresh,
} from './opertionsActions';    "opertions": Unknown word.
import {
  onFulfilledLogOut,
  onFulfilledRefresh,
  onFulfilledRegisterLogin,
  onPending,
  onRejected,
} from './helpFunctionAuth';

export const authSlice = createSlice({
  name: 'auth',
  initialState: {
    user: null,
    loading: false,
    error: null,
    showAuthForm: false,
  },
  reducers: {
    setShowAuthForm: (state, action) => {
      state.showAuthForm = action.payload;
    },
  },
  extraReducers: builder => {
    builder
      .addCase(register.fulfilled, onFulfilledRegisterLogin)
      .addCase(login.fulfilled, onFulfilledRegisterLogin)
      .addCase(loginWithGoogle.fulfilled, onFulfilledRegisterLogin)
      .addCase(logout.fulfilled, onFulfilledLogOut)
      .addCase(userRefresh.fulfilled, onFulfilledRefresh)
      .addCase(userRefresh.pending, onPending)
      .addCase(userRefresh.rejected, onRejected);
  },
});

export const { setShowAuthForm } = authSlice.actions;
```

Рис.18«Слайс» операцій аутентифікації

```
export const onPending = state => {
  state.isLoading = true;
};

export const onRejected = (state, { payload }) => {
  state.isLoading = false;
  state.error = payload;
};
export const onFulfilled = (state, { payload }) => {
  state.isLoading = false;
  state.error = '';
};

export const onFulfilledFetch = (state, { payload }) => {
  state.templates = payload.templates;
  state.favtemplates = payload.favtemplates;    "favtemplates": Unknown word.
};

export const onFulfilledAdd = (state, { payload }) => {
  state.templates.push(payload);
};
export const onFulfilledUpdate = (state, { payload }) => {
  const index = state.templates.findIndex(
    template => template.id === payload.id
  );

  if (index !== -1) {
    state.templates[index] = payload;
  }
}; ← #51-59 export const onFulfilledUpdate = (state, { payload }) =>
export const onFulfilledDelete = (state, action) => {
  state.templates = state.templates.filter(
    template => template.id !== action.payload.id
  );
}; ← #60-64 export const onFulfilledDelete = (state, action) =>

export const onFulfilledFavorite = (state, { payload }) => {
  if (payload.bool) {
    state.favTemplates.push({
      uid: payload.templateId.uid,
      id: payload.templateId.id,
    });
  } else {
    state.favTemplates = state.favTemplates.filter(
      template => template.id !== payload.templateId.id
    );
  } ← #72-76 else
}; ← #66-77 export const onFulfilledFavorite = (state, { payload }) =>
export const createStatus = type => isAnyOf( ...arrThunks.map(el => el[type]));

```

Рис. 19 Допоміжні функції на вибірку, додавання, видалення та обновлення даних у Firebase

```
import { createSlice } from '@reduxjs/toolkit';    "reduxjs": Unknown word.

import {
  STATUSES,
  createStatus,
  onFulfilled,
  onFulfilledAdd,
  onFulfilledDelete,
  onFulfilledFavorite,
  onFulfilledFetch,
  onFulfilledUpdate,
  onPending,
  onRejected,
  templatesInitialState,
} from './helpFunctionTemplate';
import {
  addTemplate,
  addTemplateFavorite,
  deleteTemplate,
  fetchTemplates,
  updateTemplate,
} from './operationsTemplate';

export const templatesSlice = createSlice({
  name: 'templates',
  initialState: templatesInitialState,
  extraReducers: builder => {
    const { PENDING, FULFILLED, REJECTED } = STATUSES;
    builder
      .addCase(fetchTemplates.fulfilled, onFulfilledFetch)
      .addCase(addTemplate.fulfilled, onFulfilledAdd)
      .addCase(updateTemplate.fulfilled, onFulfilledUpdate)
      .addCase(deleteTemplate.fulfilled, onFulfilledDelete)
      .addCase(addTemplateFavorite.fulfilled, onFulfilledFavorite)
      .addMatcher(createStatus(PENDING), onPending)
      .addMatcher(createStatus(REJECTED), onRejected)
      .addMatcher(createStatus(FULFILLED), onFulfilled);
  },
});
```

Рис. 20 «Слайс» на вибірку, додавання, видалення та обновлення даних у Firebase

У проекті використовується текстовий редактор TinyMCE для React. Це важливий елемент, який надає користувачам зручний та розширений інтерфейс для редагування текстового контенту. TinyMCE - це потужний та гнучкий текстовий редактор для веб-сайтів, що надає розширені можливості форматування та обробки тексту. Розроблений компанією Ephox, редактор став популярним завдяки своїй простоті використання та розширеній функціональності. Нижче подано огляд основних використовуваних функціональностей та інтеграції з React: Основні характеристики:

Розширений Інтерфейс та Форматування:

- TinyMCE надає багатий інтерфейс для редагування тексту, включаючи панелі інструментів для широкого спектру функцій форматування.
- Підтримка стандартних операцій, таких як жирний, курсив, підкреслення, списки, вирівнювання та інші.
- Робота з Зображеннями та Медіа:
- Можливість вставляти та редагувати зображення, вставляти відео та інші мультимедійні елементи.
- Зручний інтерфейс для роботи з атрибутами зображень, вирізання та зміна розміру.
- Розширена Таблиці та Текстові Регіони:
- Підтримка таблиць для організації табличної інформації.
- Можливість вставки та форматування текстових регіонів, що дозволяє розміщувати текст у визначені області.
- Підтримка Мов та Локалізація:
- Підтримка різних мов та можливість налаштування локалізації.
- Легкість інтеграції з різними мовами для задоволення потреб різних аудиторій.

Плагіни та Розширення:

- Можливість розширювати функціональність редактора за допомогою плагінів.
- Наявність ряду стандартних плагінів для різноманітних задач.
- Адаптивний та Сумісний:
- Адаптований для використання на різних пристроях та екранах.
- Сумісний з багатьма сучасними браузерами, що робить його доступним для широкого кола користувачів.

Кросплатформений та Відкритий Код:

Підтримка великої кількості платформ, включаючи розгортання на різних операційних системах.

Відкритий код, що дозволяє розробникам вносити власні виправлення та розширювати функціонал.

Загалом, використання TinyMCE в вашому проекті додає значну функціональність та зручність для користувачів, які мають можливість редагувати текстовий контент у простий та ефективний спосіб[див. рис 21].

```
import React from 'react';
import { Editor } from '@tinymce/tinymce-react';    "tinymce": Unknown word.

const MyEditor = ({ editorValue, handleChange }) => {
  return (
    <Editor
      initialValue={editorValue}
      apiKey="a85ckh1rvs1kxnitvo54z16evw4mvc00gqs3ukonw5rzvgd1"
      init={{
        height: '100%',
        menubar: true,

        branding: false,
        plugins: [
          'advlist autolink lists link image charmap print preview anchor',    "advlist": Unk
          'searchreplace visualblocks code fullscreen',    "searchreplace": Unknown word.
          'insertdatetime media table paste code help wordcount',    "insertdatetime": Unknow
        ], ← #14-18 plugins:
        toolbar:
          'undo redo | formatselect | ' +    "formatselect": Unknown word.
          'bold italic backcolor | alignleft aligncenter ' +    "backcolor": Unknown word.
          'alignright alignjustify | bullist numlist outdent indent | ' +    "alignright": Un
          'removeformat | help',    "removeformat": Unknown word.
        // Додаткові опції    "Додаткові": Unknown word.
        fontsize_formats: '8pt 10pt 12pt 14pt 18pt 24pt 36pt',
        language: 'uk',
        content_style: 'body { font-family: Arial, sans-serif; }',
        automatic_uploads: true,
        file_picker_types: 'image',
        file_picker_callback: (callback, value, meta) => {
          // Додайте свій власний код для вибору файлів    "Додайте": Unknown word.
        },
        // Інші опції за потреби    "Інші": Unknown word.
      }} ← #9-34 init=
      onEditorChange={handleChange}
    />
  ); ← #5-37 return
}; ← #4-38 const MyEditor = ({ editorValue, handleChange }) =>

export default MyEditor;
```

Рис. 21 Код редактору шаблону

TinyMCE надає можливість користувачу роздруковувати створені ним шаблон.

Також у системі є можливість загружати шаблони та редагувати їх. Для перетворення між різними форматами використовуються бібліотеки convertapi-js та html2pdf.

Ці дві бібліотеки спільно допомагають забезпечити проект засобами для ефективного створення та конвертації документів, роблячи розробку звітів, матеріалів чи інших важливих вмістів більш зручною та продуктивною.

html2pdf.js – це потужний інструмент для генерації PDF-файлів на основі вмісту веб-сторінок. Забезпечуючи можливість конвертувати HTML та CSS в PDF, ця бібліотека стає корисним рішенням для створення документів, звітів чи інших матеріалів зі структурованим вмістом. Зокрема, вона дозволяє деталізовано керувати налаштуваннями PDF, такими як розміри, якість зображень та інші параметри.

Бібліотека html2pdf може бути використана для зручного створення PDF-файлів з вмісту, що вже присутній в React-додатках. За допомогою інтеграції цієї бібліотеки можна генерувати PDF з обраних елементів або сторінок, забезпечуючи при цьому розширені можливості налаштувань для кожного PDF-документа[див. рис 22].

```
const handleGeneratePDF = () => {
  const content = editorValue;

  html2pdf(content, {
    margin: 10,
    filename: 'document.pdf',
    image: { type: 'jpeg', quality: 0.98 },
    html2canvas: { scale: 2 },
    jsPDF: { unit: 'mm', format: 'a4', orientation: 'portrait' },
  }); ← #103-109 html2pdf
}; ← #100-110 const handleGeneratePDF = () =>
```

Рис. 22 Використання html2pdf

convertapi-js – це JavaScript SDK для роботи з сервісом ConvertAPI, який спеціалізується на конвертації різних типів файлів в інші формати. У контексті даного проекту, використовується для здійснення конвертації документів з формату docx в формати html та jpg.

Бібліотека convertapi-js дозволяє ефективно виконувати конвертацію документів у форматі docx у різні інші формати. Вона інтегрується з React-додатками для обробки файлів та отримання результатів конвертації, забезпечуючи при цьому зручний та надійний інтерфейс для роботи з ConvertAPI[див. рис 23].

```
const handleFileChange = async event => {
  const file = event.target.files[0];
  setTemplateName(file.name);
  const params = convertApi.createParams();
  params.add('file', file);

  try {
    const result = await convertApi.convert('docx', 'html', params);
    const result1 = await convertApi.convert('docx', 'jpg', params);
    // Отримайте посилання на завантаження HTML-файлу    "Отримайте": Unknown word.
    const htmlFileUrl = result.files[0].Url;
    const htmlFileUrl1 = result1.files[0].Url;

    const prevUrl = await setImage(
      'previous',
      htmlFileUrl1,
      result1.files[0].FileId
    ); ← #65-69 const prevUrl = await setImage
    // Завантажте вміст HTML-файлу    "Завантажте": Unknown word.
    setTemplateImageUrl(prevUrl);
    const response = await fetch(htmlFileUrl);
    const content = await response.text();

    setEditorValue(content);
  } catch (error) {
    console.error('Error converting Word file:', error);
  }
}; ← #52-79 const handleFileChange = async event =>
```

Рис. 23 Використання convertapi-js

Для нереєстрованих користувачів у проекті забезпечено зручну можливість переглядати зображення шаблонів з водяними знаками. Це дає можливість ознайомитися з доступними шаблонами та отримати уявлення про їх вміст, надаючи користувачам можливість вибору та оцінки шаблонів навіть до реєстрації. Водяні знаки ефективно захищають авторські права та властивості, забезпечуючи відповідний контроль над зображеннями. Такий підхід сприяє позитивному користувацькому досвіду, дозволяючи користувачам отримати приємні враження від шаблонів та взаємодії з проектом [див. рис 24].

```
export const PdfBox = styled.div`
  margin: 0 auto;
  width: 50%;
  height: 100%;
  background-image: url(${SiImage}), url(${props => props.url});
  background-position: center, center;
  background-size: 50px, contain;
  background-repeat: space, no-repeat;
`;
```

Рис. 24 Код шаблонів з водяними знаками для незареєстрованих користувачів

Компонент TemplateList відповідає за відображення списку шаблонів у вашому веб-додатку. Розглянемо його функціонал та основні елементи:

Компонент імпортує необхідні бібліотеки та стилі для оформлення. Також використовуються іконки для різних дій.

Компонент приймає пропс allTemplates, який містить масив інформації про усі шаблони, які треба відобразити. Крім того, використовуються функції з Redux для взаємодії зі станом додатку.

Видалення шаблону: При кліці на кнопку видалення викликається функція handleDelete(id), яка диспетчеризує Redux-екшен для видалення шаблону.

Додавання до обраних: При кліці на кнопку "Додати до обраних" викликається функція handleFavorite(templateId, bool), яка додає або видаляє шаблон з обраних за допомогою Redux-екшену.

Редагування шаблону: Клік на кнопку редагування перенаправляє користувача на сторінку редагування відповідного шаблону.

Відображення інформації:

Кожен елемент TransportationItem включає в себе ім'я шаблону (P) та зображення шаблону, яке є посиланням для переходу на сторінку редагування.

Стилі компонента задаються через власні стилі TemplatesList.styled. Наприклад, використовуються стилі для кнопок редагування, видалення та додавання до обраних.

Загалом, TemplateList забезпечує зручний інтерфейс для перегляду та взаємодії зі списком шаблонів, дозволяючи користувачам здійснювати різні дії з кожним шаблоном[див. рис 25].

```
} from 'redux/templates/operationsTemplate';

const TemplateList = ({ allTemplates, onButton }) => {
  const user = useSelector(selectUser);
  const favTemplates = useSelector(selectFavTemplates);

  const dispatch = useDispatch();

  const handleDelete = id => {
    dispatch(deleteTemplate(id));
  };
  const handleFavorite = (templateId, bool) => {
    console.log('templateId, bool :>> ', templateId, bool);
    dispatch(addTemplateFavorite({ templateId, bool }));
  };
  return (
```

Рис. 25 Код компонента TemplateList

Компонент TemplateFind відповідає за відображення блоку пошуку та фільтрації шаблонів у веб-додатку. Розглянемо його основні елементи та функціонал:

Компонент є функціональним та використовує хук стану useState для зберігання значення терміну пошуку (searchTerm).

Input: Введене користувачем значення терміну пошуку зберігається у стані компонента.

handleInputChange: Функція викликається при зміні введеного значення і оновлює стан searchTerm.

Кнопка "Пошук" (handleSearchClick): При кліці на цю кнопку викликається функція onSearch, яка передає термін пошуку для подальшої обробки.

Кнопка "Обрані" (handleFavoriteClick): При кліці на цю кнопку викликається функція onFavorite, яка відповідає за відображення обраних шаблонів або їх приховування.

Стилі компонента задаються через власні стилі TemplateFind.styled. Наприклад, використовуються стилі для введення, кнопок та контейнера пошуку.

Загалом, TemplateFind дозволяє користувачеві вводити терміни пошуку та взаємодіяти з функціоналом пошуку та фільтрації шаблонів у вашому веб-додатку[див. рис 26].

```
import React, { useState } from 'react';
import { Button, FindContainer, Input } from './TemplateFind.styled';

const TemplateFind = ({ onSearch, onFavorite }) => {
  const [searchTerm, setSearchTerm] = useState('');

  const handleInputChange = event => {
    setSearchTerm(event.target.value);
  };

  const handleSearchClick = () => {
    onSearch(searchTerm);
  };
  const handleFavoriteClick = () => {
    onFavorite(prev => (prev === 'true' ? '' : 'true'));
  };
  return (
    <FindContainer>
      <Input
        type="text"
        placeholder="Введіть термін пошуку"    "Введіть": Unknown word.
        value={searchTerm}
        onChange={handleInputChange}
      />
      <Button onClick={handleSearchClick}>Пошук</Button>    "Пошук": Unknown word.
      <Button onClick={handleFavoriteClick}>Обрані</Button>    "Обрані": Unknown word.
    </FindContainer>
  ); ← #17-28 return
}; ← #4-29 const TemplateFind = ({ onSearch, onFavorite }) =>

export default TemplateFind;
```

Рис. 26 Код компонента TemplateFind

4.2. Тестування веб-сайту інтелектуальної ситеми створення шаблонів підприємця. «AI DOCS»

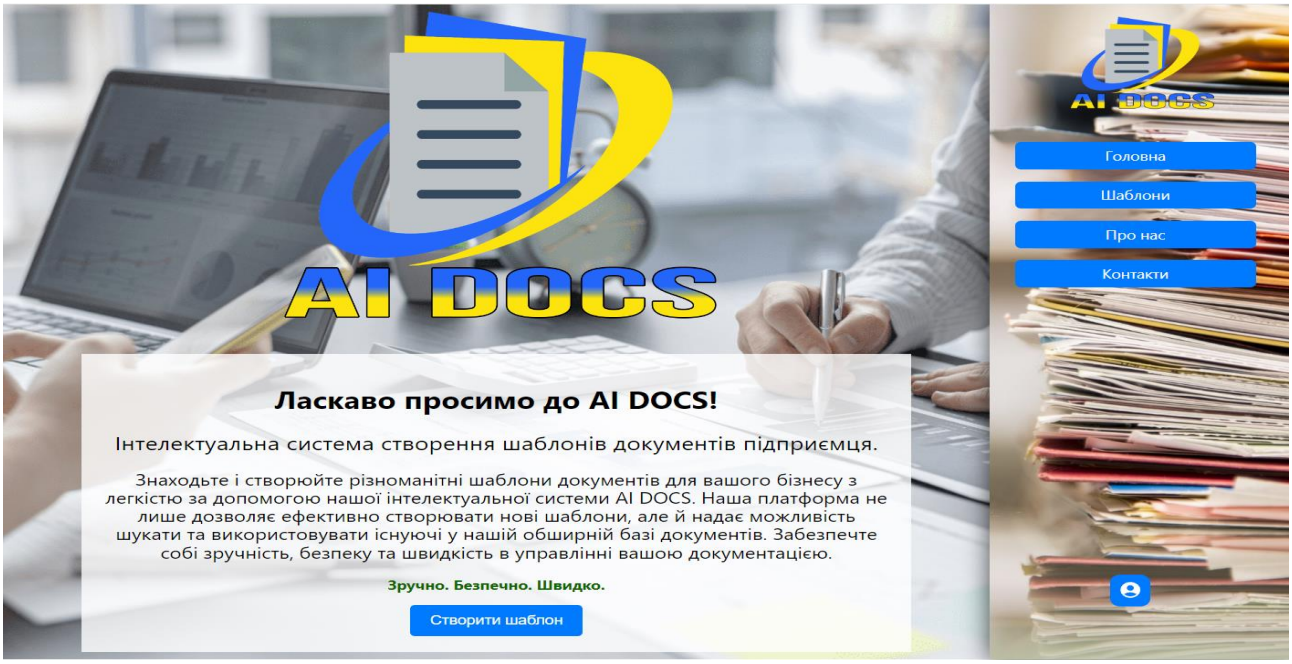


Рис. 27. Головна сторінка

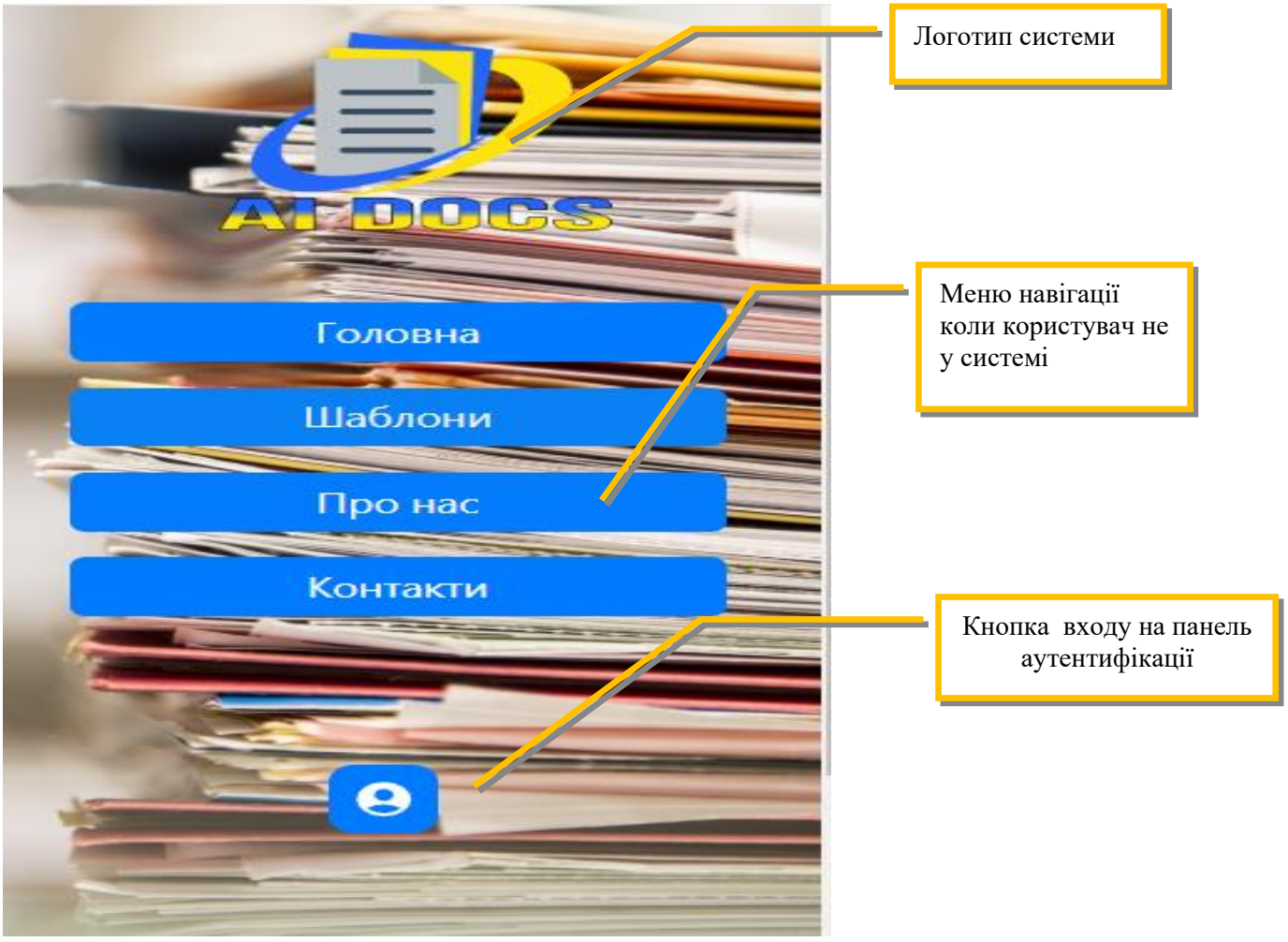


Рис. 28. Бічна панель керування з навігацією та кнопкою аутентифікації.

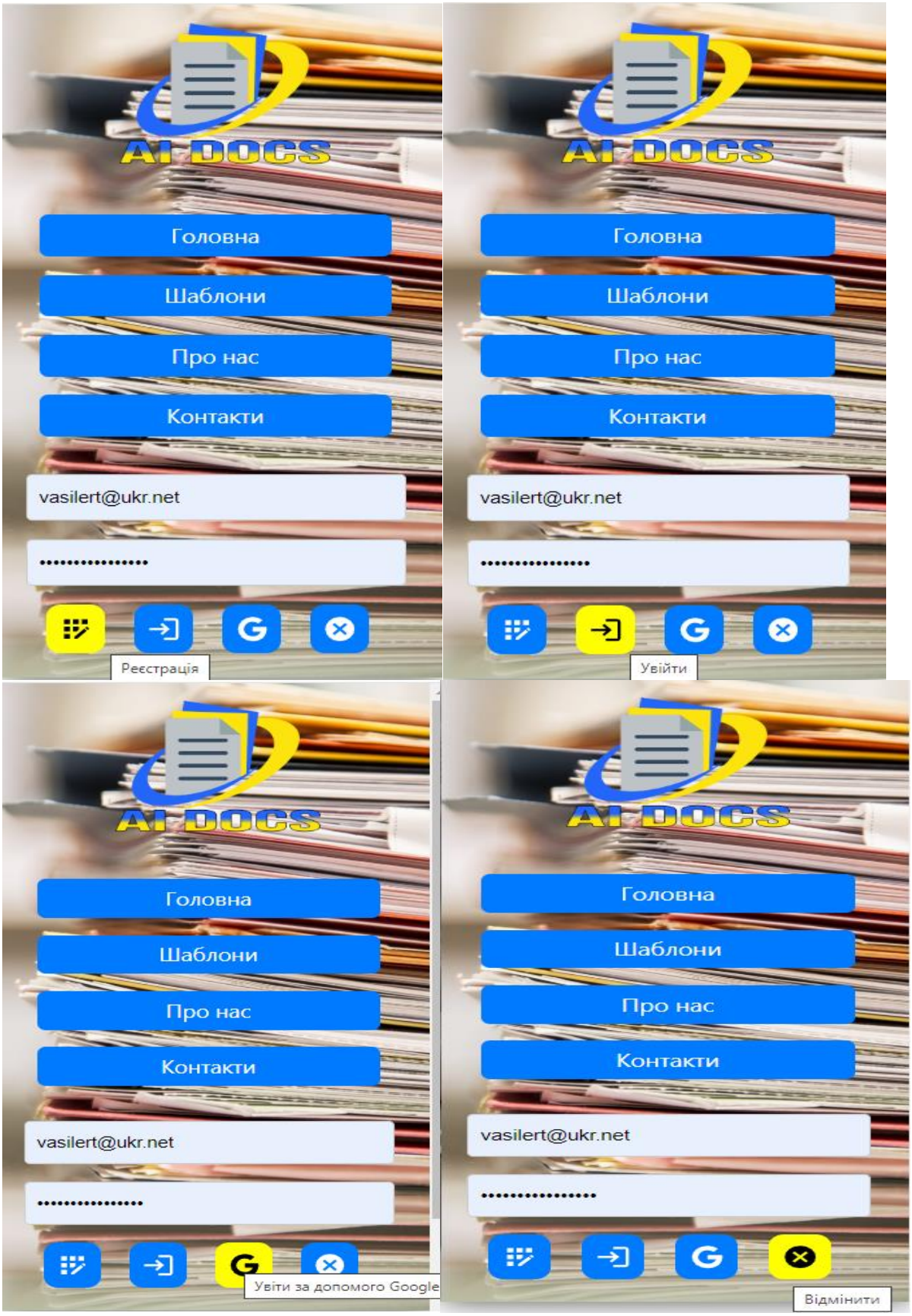


Рис. 29. Елементи керування панелі аутентифікації .

Рис. 30. Сторінка «Про нас».

Рис. 31. Сторінка «Контакти».

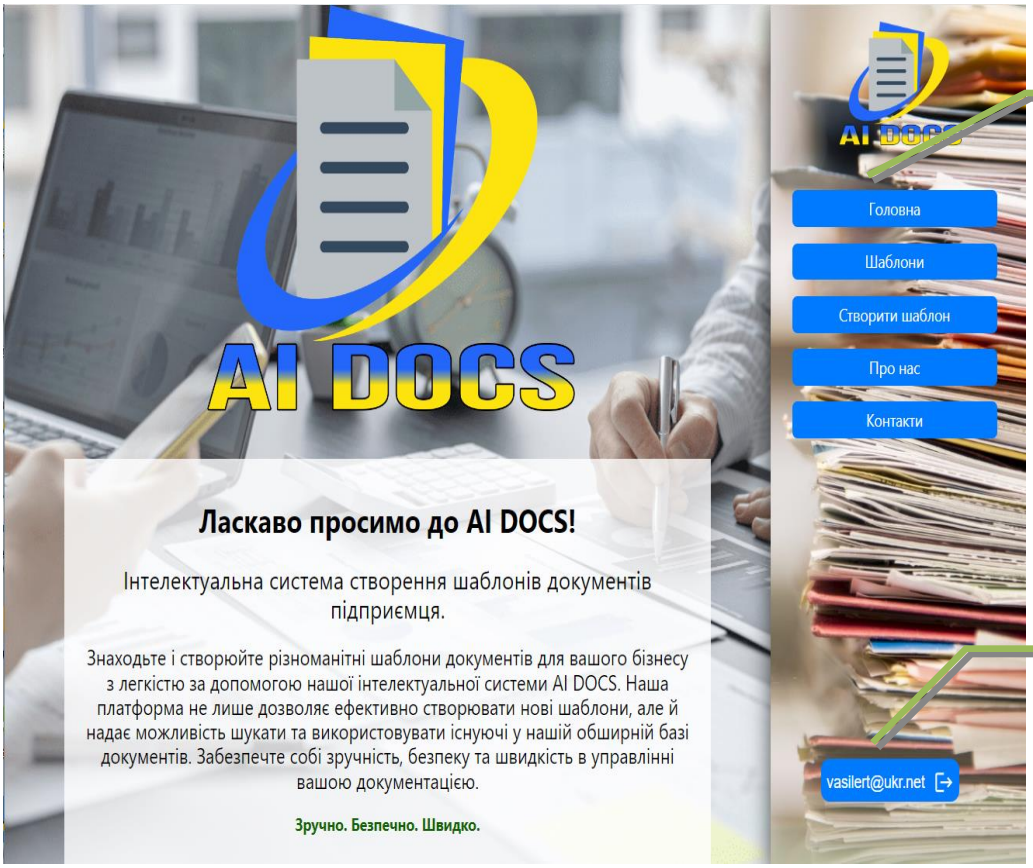


Рис. 32. Розширене Навігаційне меню коли користувач увійшов в систему

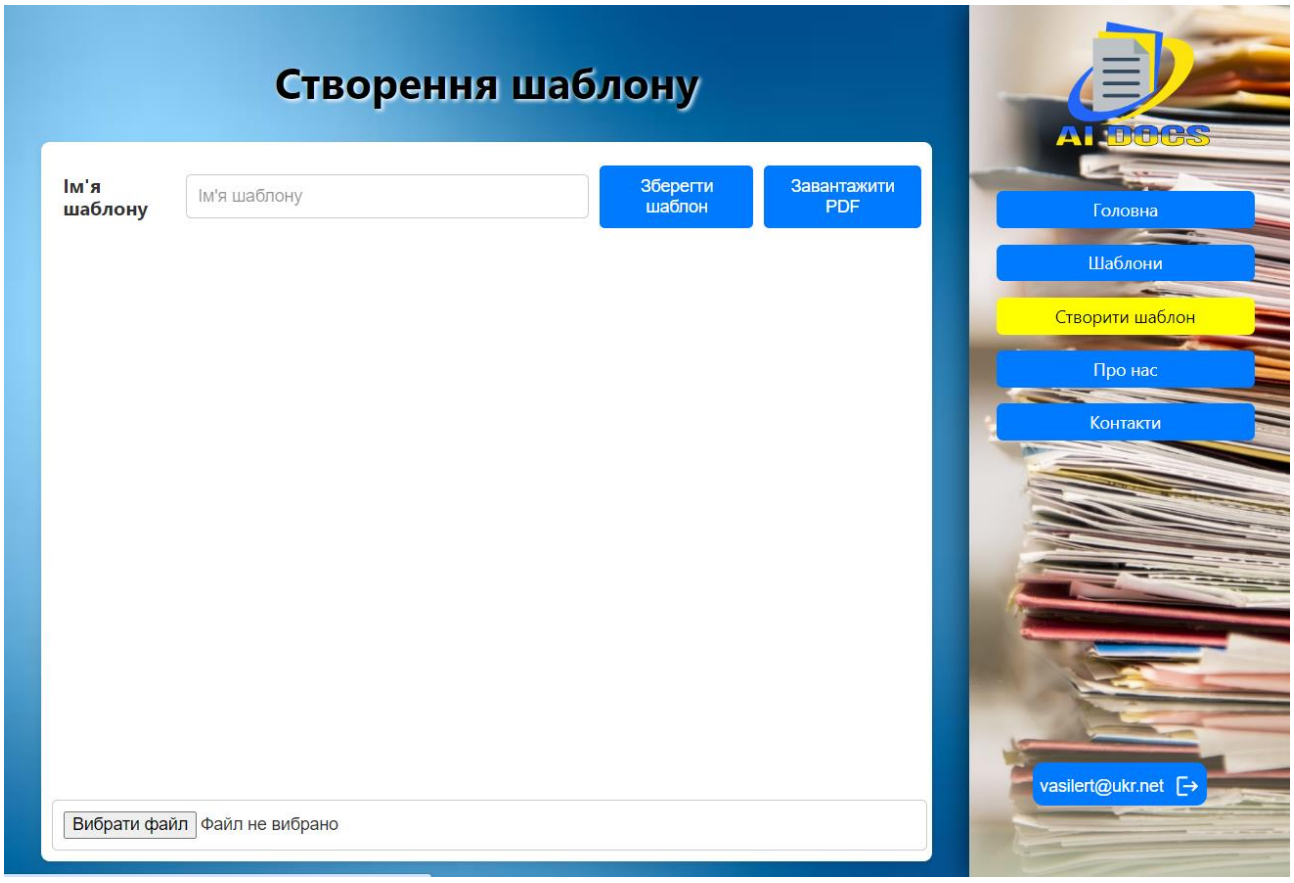


Рис. 33. Сторінка створення шаблона

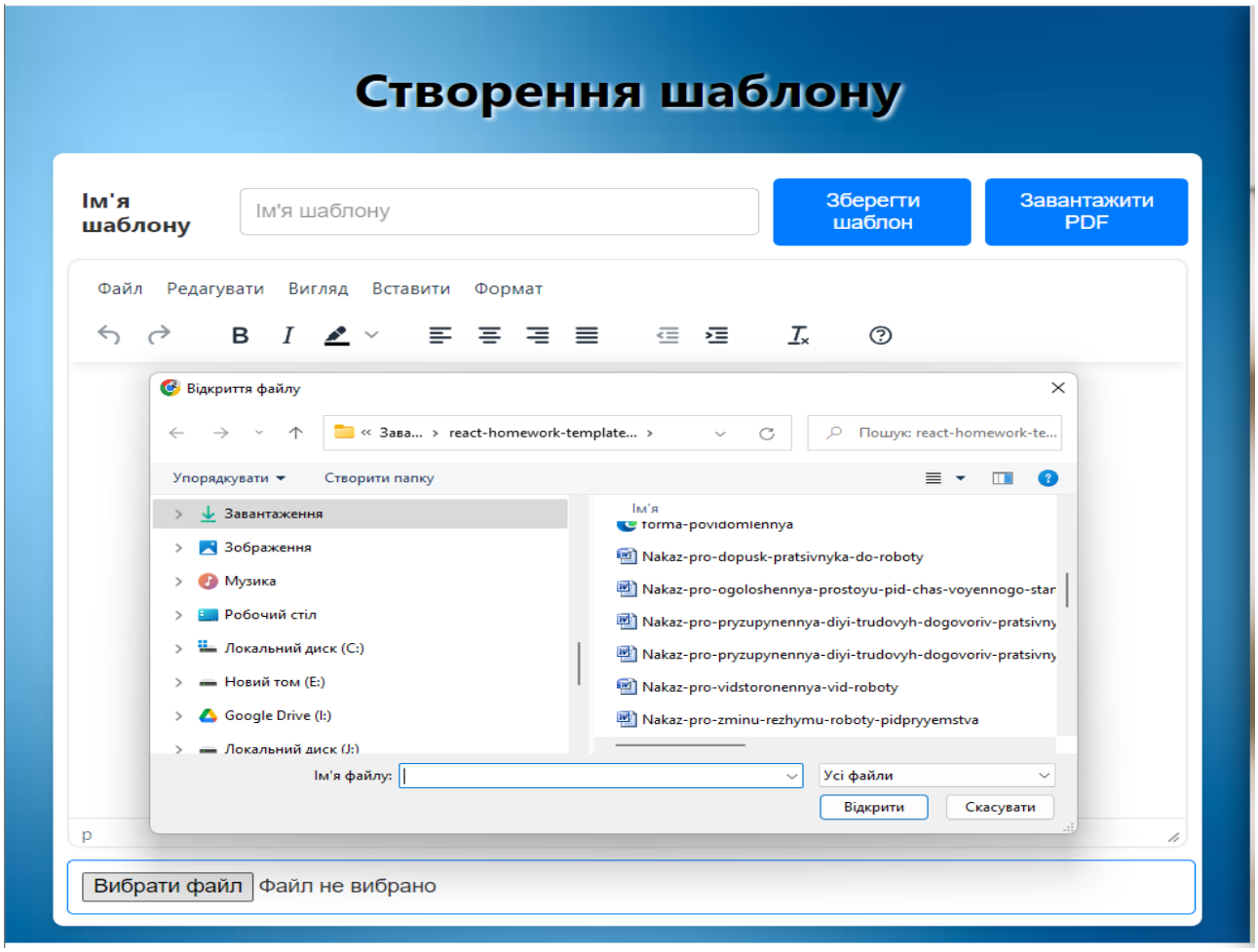


Рис. 34. Завантаження шаблону користувача для редагування

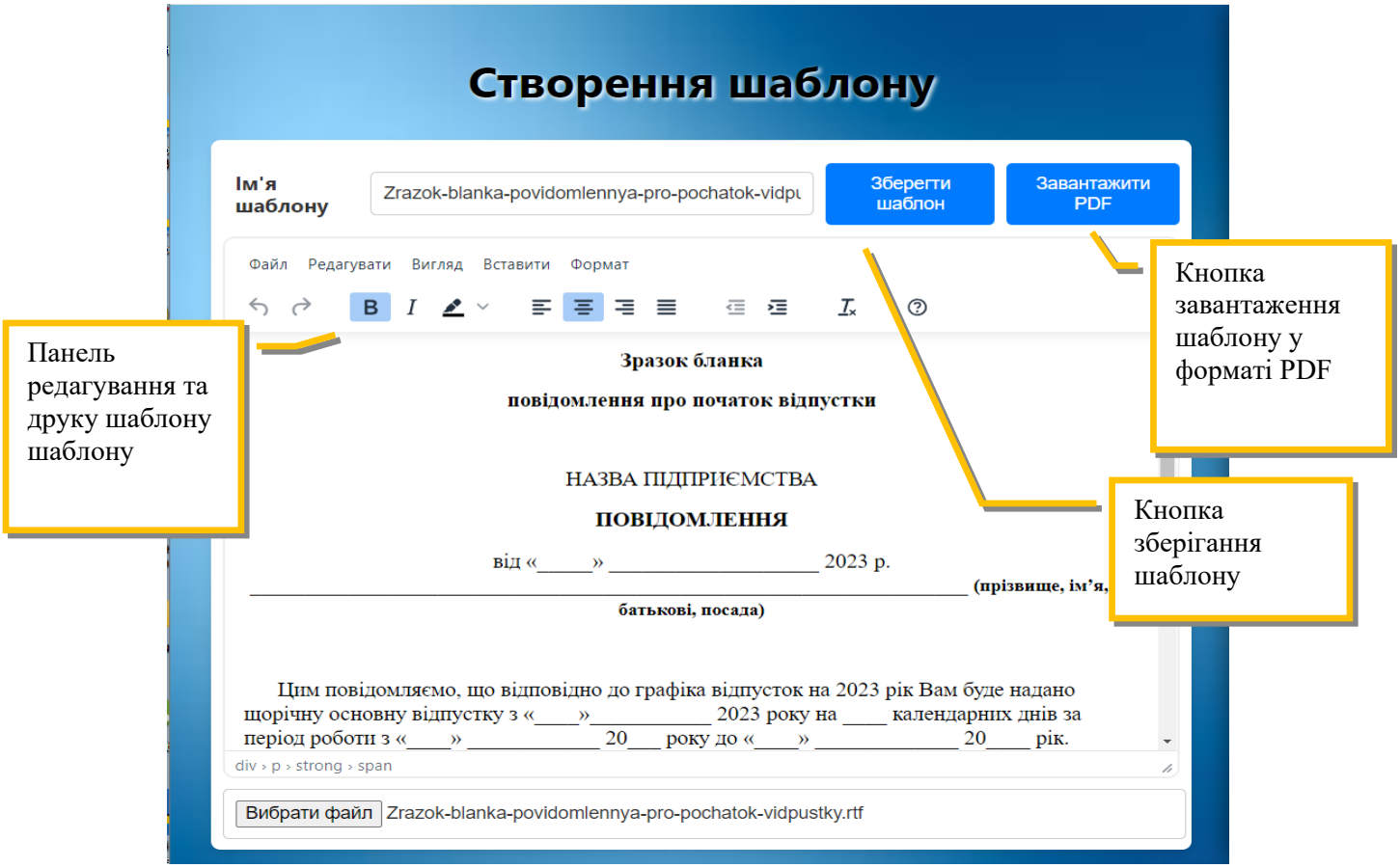


Рис. 35. Редагування та збереження шаблону

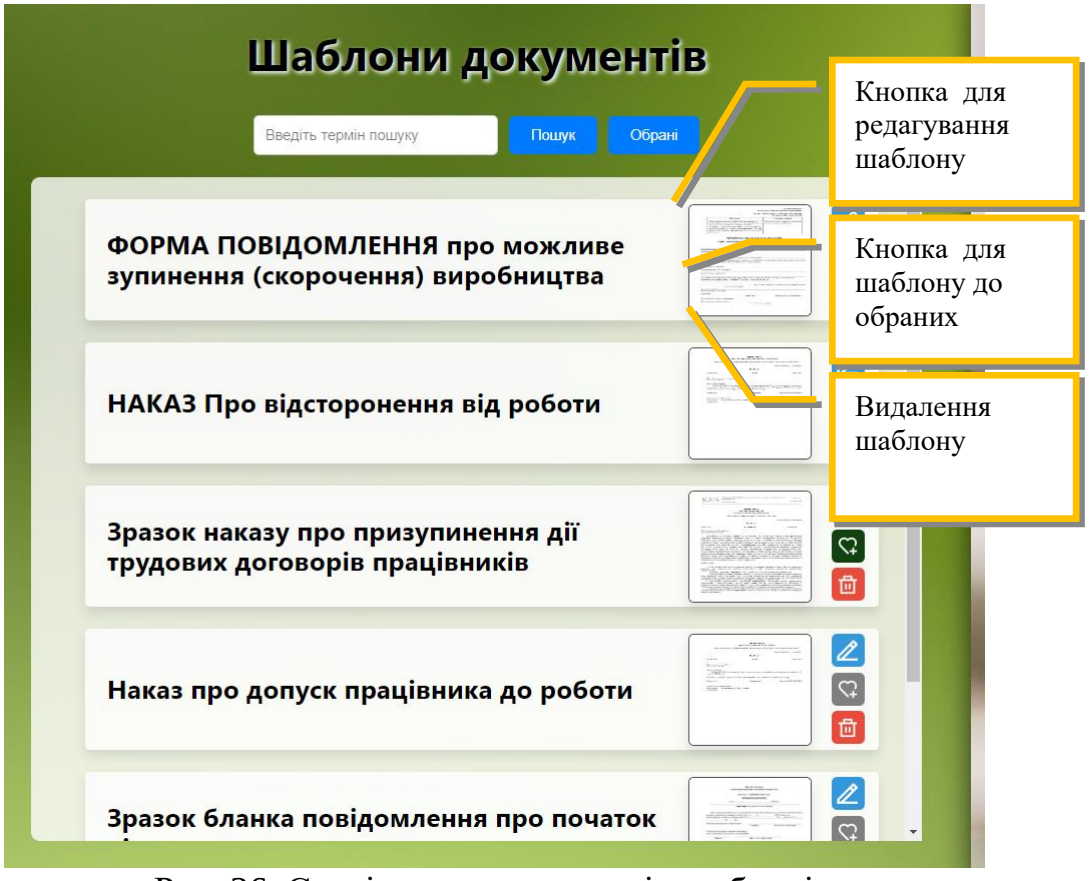


Рис. 36. Сторінка перегляду всіх шаблонів

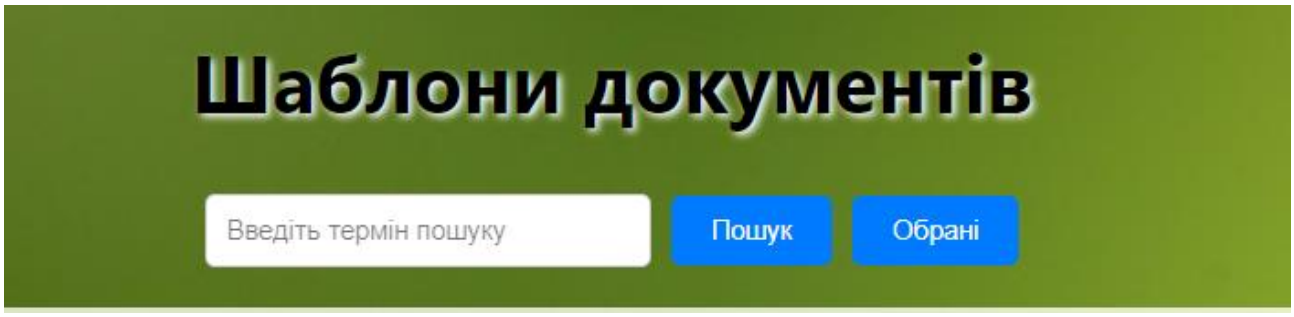


Рис. 37. Панель пошуку

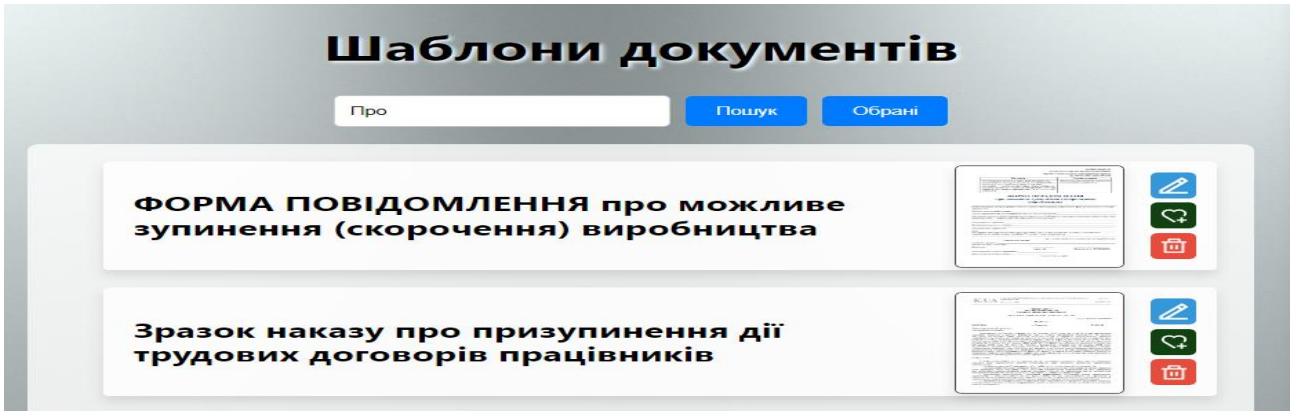


Рис. 38. Обрані шаблони

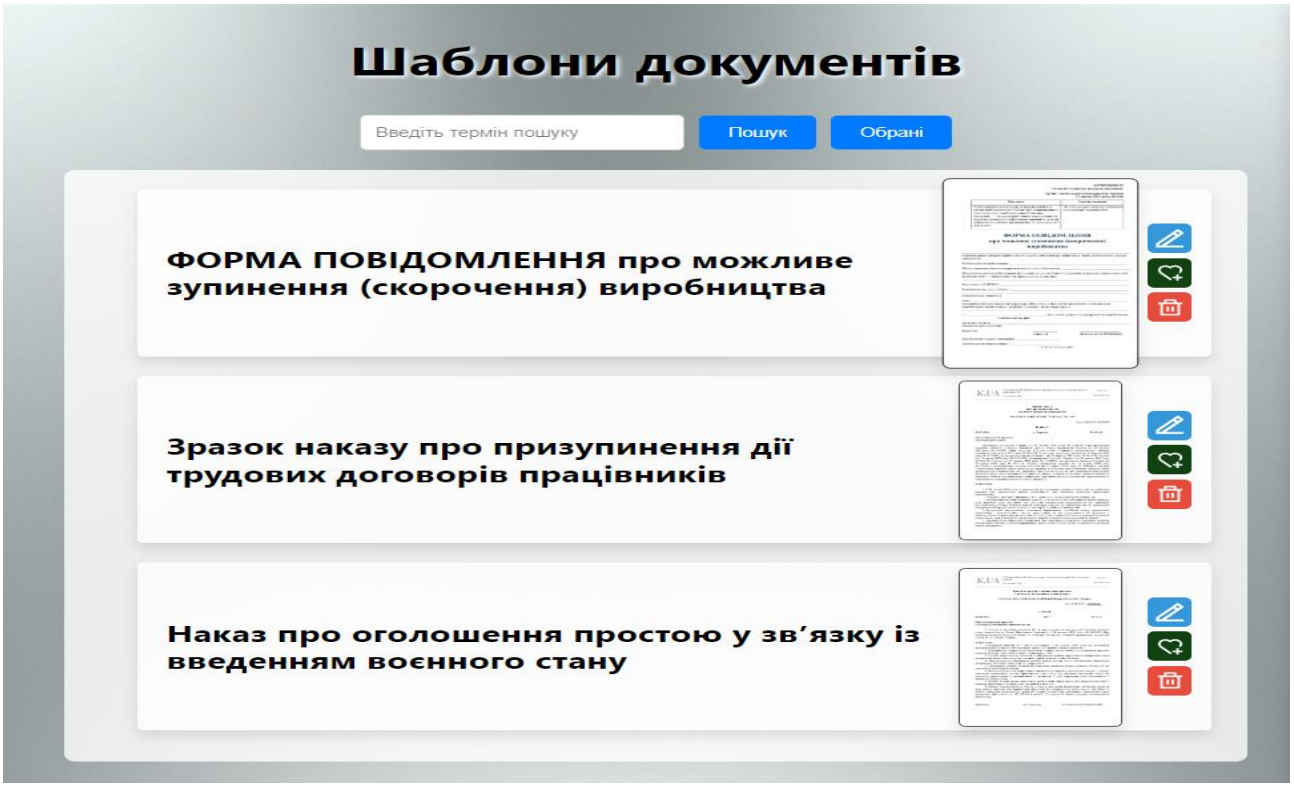


Рис. 39. Обрані шаблони користувача

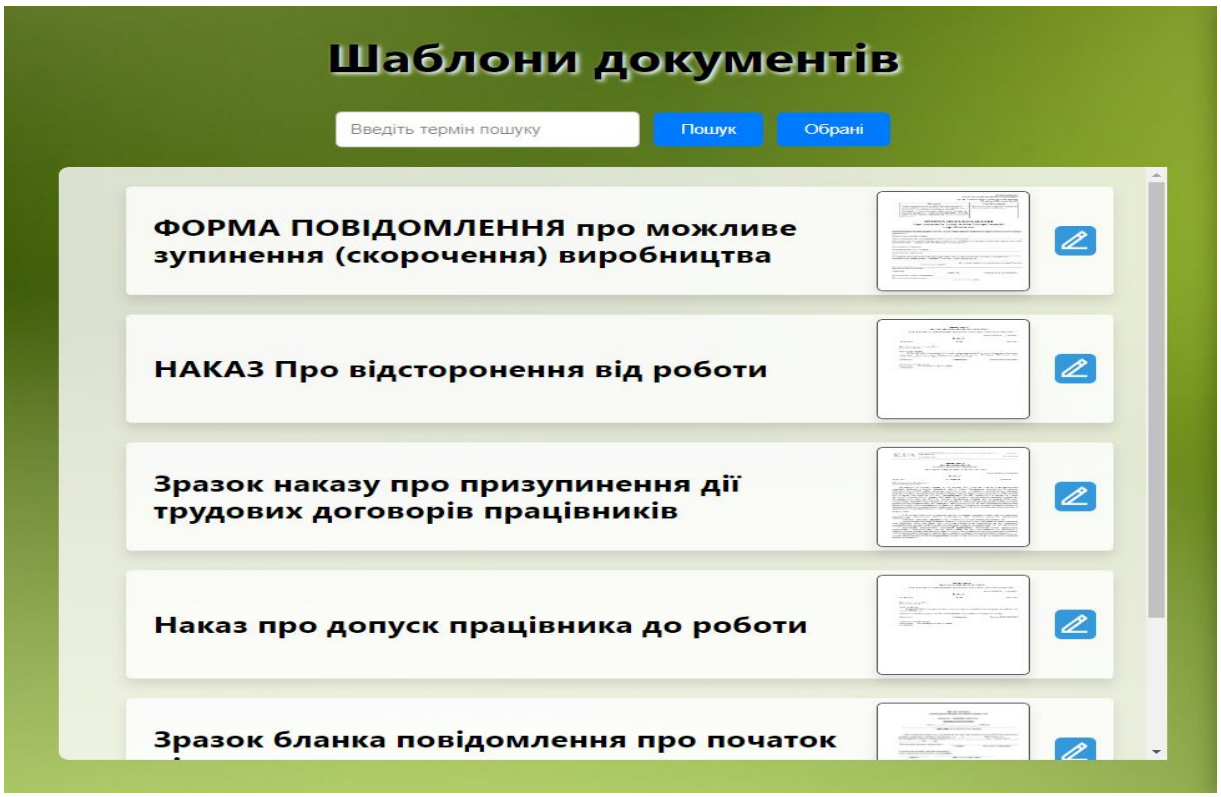


Рис. 40. Шаблони для нереєстрованого користувача

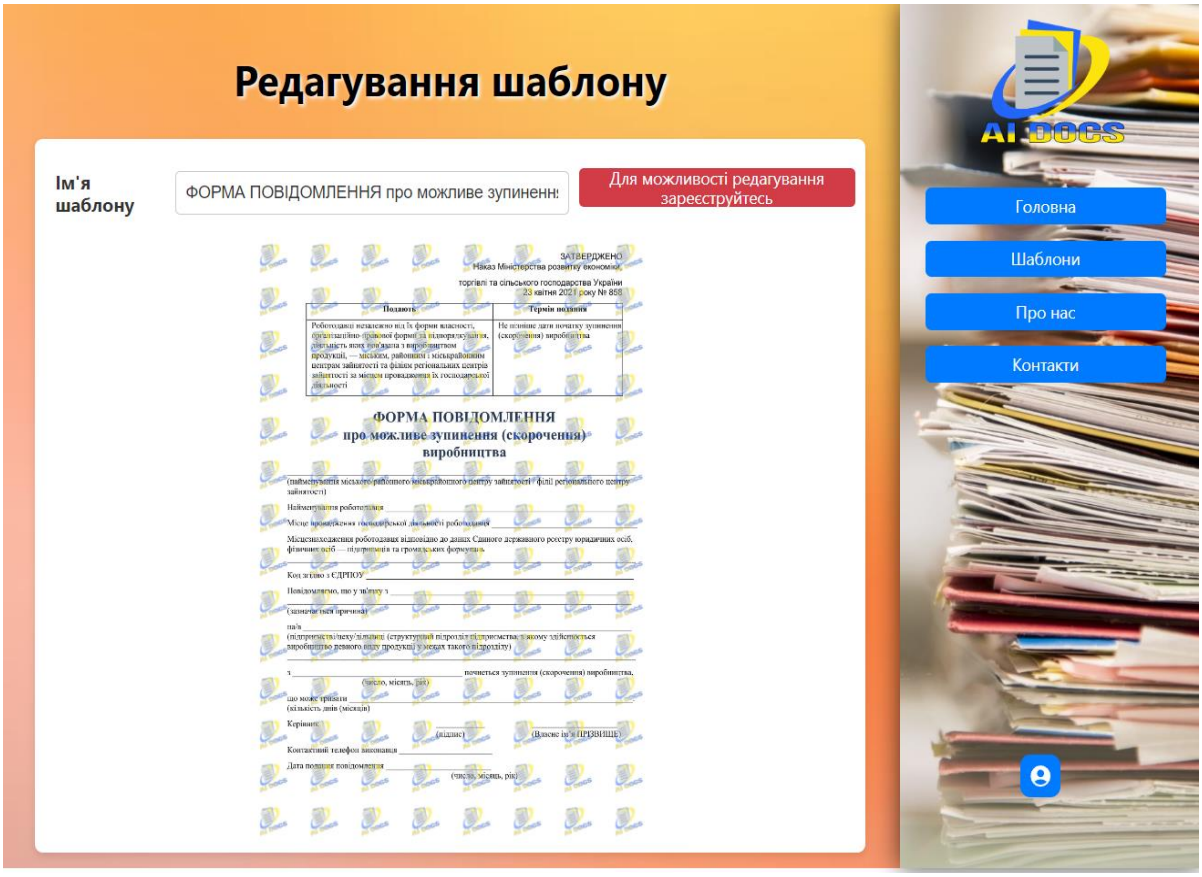


Рис. 41. Редактор шаблонів для нереєстрованого користувача

4.3. Висновки до розділу

AiDocs є інструментом, який використовує сучасні підходи до розробки, зокрема, React, для створення інноваційних рішень у сфері управління діловими документами.

React надає гнучкість у створенні повторно використовуваних компонентів, тим самим дозволяє будувати складні структури веб-сайтів з простих блоків. А це в свою чергу спрощує розробку та підтримку коду, оскільки всі компоненти можна легко модифікувати та використовувати в різних частинах системи.

Крім того, використання функціональних компонентів та хуків у React, таких як useState і useEffect, дозволяє ефективно управляти станом додатка та реагувати на події в реальному часі. Це дозволяє створювати інтерактивні та реактивні елементи в платформі AiDocs, що сприяє зручності користувача та покращує взаємодію з документами.

Такий підхід до розробки дозволяє створювати динамічні, легко змінювані та реагуючі елементи платформи, що може значно полегшити процес створення та управління діловими паперами.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП ПРОЕКТУ

5.1. Опис ідеї проекту

Електронний документообіг є критично важливим для ефективності управління документами та робочих процесів в сучасних установах. Оптимізація роботи через ці системи може значно полегшити процес обробки та доступ до інформації. Інтеграція електронної пошти в цю систему - це крок у напрямку зменшення рутинних завдань, відкриваючи можливості для більш ефективного використання часу та ресурсів.

Щодо великого обсягу документів, що формуються регулярно, автоматизовані системи можуть допомогти з відстеженням строків, нагадуванням про них та автоматичним призначенням завдань для їх обробки. Це дозволить уникнути затримок та забезпечити вчасне виконання завдань.

Інтеграція функцій аналізу даних та звітності також важлива. Це дозволяє зібрати та аналізувати дані про документообіг, що допомагає вдосконалити процеси, виявити можливі проблеми та зробити зрозумілішою внутрішню роботу.

В цілому, ефективне використання електронного документообігу сприяє покращенню продуктивності та забезпеченню швидкого та ефективного доступу до важливої інформації.

5.2. Аналіз технологічних можливостей реалізації проєкту " AiDocs "

Система AiDocs, як потужний інструмент для створення та управління документами. Його можливість швидко створювати шаблони на основі існуючих - велика перевага для користувачів, оскільки це спрощує і прискорює процес створення документів. Також, доступ до великої бази шаблонів дозволяє користувачам знайти оптимальний варіант для своїх потреб та налаштувати його за бажанням. Функціонал друку та завантаження документів у форматі PDF надає зручність та можливість зберігати або надсилати документи в потрібному форматі. Забезпечення цих функцій для зареєстрованих користувачів сприяє захисту конфіденційності та безпеці інформації.

Також, можливість переглядати захищені водяними знаками зображення шаблонів для незареєстрованих користувачів - це чудовий спосіб дати їм уявлення про функціонал програми перед реєстрацією. Використання фреймворку React додає реактивності та зручності у користуванні системою. Компонентний підхід дозволяє з легкістю взаємодіяти з різними функціями системи, що може покращити загальний досвід користувача та забезпечити більшу ефективність використання програми.

Використання React у AiDocs має дійсно багато переваг, особливо завдяки використанню віртуального DOM. Це дозволяє системі швидко та ефективно відображати зміни в інтерфейсі, при цьому зменшуючи навантаження на браузер користувача. Такий підхід забезпечує високу продуктивність додатку.

Принцип одностороннього потоку даних у AiDocs сприяє ефективному відстеженню та керуванню станом компонентів. Це допомагає забезпечити послідовність та передбачуваність роботи системи і полегшує виявлення та виправлення помилок.

Розширюваність React разом із додатковими бібліотеками, такими як Redux та React Router, дозволяє AiDocs легко адаптуватися та розширюватися для надання більших можливостей та функціональності. Активне співтовариство розробників важливе, оскільки воно забезпечує доступ до багатьох ресурсів, підтримки та документації, що робить цей інструмент ще потужнішим і допомагає вирішувати складні завдання.

5.3. Аналіз ринкових можливостей запуску стартап-проєкту

Врахування користувацьких потреб - ключовий фактор успіху для будь-якого стартапу, включаючи AiDocs. Залучення користувачів та створення продукту, що відповідає їхнім потребам, допомагає забезпечити успішний запуск та подальший розвиток. Мобільні рішення сьогодні справді є важливими для широкого кола користувачів, тому мають великий потенціал у ринковій конкуренції. AiDocs,

задовольняючи потреби в організації роботи з документами, може привернути різні професійні, соціальні та вікові групи.

Створення популярного додатку - це не просто про функціональність. Важливо, щоб інтерфейс був зручним, легким у використанні та відповідав потребам різних груп користувачів. Це означає, що він повинен бути інтуїтивно зрозумілим, ефективним та привабливим для різних категорій людей. Залучення широкої аудиторії - це не лише стратегічний аспект, а й ключ до популярності та стійкого розвитку. Якщо AiDocs зможе задовольнити потреби різноманітних користувачів, він може здобути велику популярність та успіх на ринку мобільних рішень.

5.4. Розроблення ринкової стратегії проєкту

Враховуючи те, що даний застосунок працюватиме на усі категорії користувачів слід застосовувати масовий маркетинг, пропонуючи стандартизовану програму:

1. Провівши аналіз ринку, поточні ціни, моделі монетизації, плюси та мінуси схожих додатків конкурентів, створений "e-mail Hub" вирізняється своєю практичністю, якістю та доступністю.
2. Заплановано створення веб-сайту, цільової сторінки або тизерного відео. Оскільки, такі цільові сторінки є джерелом встановлення додатків. Окрім створення ажіотажу навколо застосунку, наявність веб-сайту перед запуском також є чудовим способом провести ранню пошукову оптимізацію (SEO), почавши нарощувати авторитет домену.
3. Просування в соціальних мережах Facebook, Instagram, Twitter, LinkedIn, Pinterest та інших соціальні платформах дозволить розширити присутність проєкту в Інтернеті та напряму спілкуватися з цільовою аудиторією. Збільшуємо впізнаваність застосунку за допомогою контенту. Намагаємось створити бренд навколо "e-mail Hub", ставши експертом у даній ніші.

5.5. Висновки розділу

AiDocs та його успішність свідчать про актуальність нових підходів у розробці програмного забезпечення. Цей застосунок показує, що користувачі цінують зручність, швидкість та функціональність у роботі з документами. Результати аналізу ринку також підтверджують, що попит на такі інноваційні інструменти постійно зростає.

Це свідчить про необхідність постійного вдосконалення розробки програмного забезпечення та уважного відношення до потреб користувачів. Інновації у цій області можуть включати нові підходи до взаємодії з програмами, покращення ефективності та надійності систем, а також вдосконалення інтерфейсів для більшого комфорту користувачів. Створення AiDocs та аналіз ринку наочно підтверджують, що розвиток програмного забезпечення вимагає постійних зусиль у напрямку інновацій та адаптації до змінних потреб користувачів.

ВИСНОВКИ

Відповідно до вимог технічного завдання, розроблено програму AiDocs, яка поєднує в собі зручність, швидкість та функціональність для створення різноманітної документації. Застосунок виступає не лише, як засіб для створення документів, але й як повноцінна платформа з необхідними інструментами для продуктивної роботи з документацією.

Компонентний підхід, на якому базується AiDocs, вказує на те, що система побудована у такий спосіб, дозволяє розділити її на взаємозалежні, але автономні частини. Це сприяє зручності розробки та модифікації, що може значно полегшити процес внесення змін та додавання нового функціоналу.

Такий підхід дійсно може допомогти у підтримці якості та функціональності системи у майбутньому, оскільки його структура сприяє зручності роботи та модифікації, а також дозволяє ефективно використовувати компоненти для покращення та розвитку системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Machining Fundamentals Tenth Edition, by John R. Walker , Bob Dixon. January 25, 2018
2. CNC Programming Handbook, Third Edition by Peter Smid, November 26, 2007
3. GD&T: Application and Interpretation Seventh Edition, Revised, by Bruce A. Wilson, November 1, 2019
4. Анісімов В.А., Терещенко В.М., Кравченко І.В. Основні алгоритми обчислювальної геометрії: Навч. посібн. – К.: Київський університет, 2002.
5. http://graphics.cs.ucdavis.edu – сайт з КГ інституту аналізу даних і візуалізації Каліфорнійського університету.
6. http://www.cg.tuwien.ac.at/courses/cg2 – сайт Інституту комп’ютерної графіки і алгоритмів Віденського технічного університету.

ДОДАТКИ

Сторінка «index.js»

index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { App } from 'components/App';
import './index.css';
import { BrowserRouter } from 'react-router-dom';
import { Provider } from 'react-redux';

import { PersistGate } from 'redux-persist/integration/react';
import { persistor, store } from 'redux/store';    "persistor": Unknown word.

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <BrowserRouter basename="/ai-docs">
      <Provider store={store}>
        <PersistGate loading={null} persistor={persistor}>    "persistor": Unknown word.
          <App />
        </PersistGate>
      </Provider>
    </BrowserRouter>
  </React.StrictMode>
); ← #11-21 ReactDOM.createRoot(document.getElementById('root')).render

```

Компонент «App.jsx»

```
import Home from 'pages/Home/Home';
import { Route, Routes } from 'react-router-dom';
import Layout from './Layout/Layout';
import About from 'pages/About/About';
import Contacts from 'pages/Contacts/Contacts';
import Templates from 'pages/Templates/Templates';

import TemplateEditor from 'pages/TemplateEditor/TemplateEditor';

export const App = () => {
  return (
    <Routes>
      <Route path="/" element={<Layout />}>
        <Route index element={<Home />} />
        <Route path="template-edit" element={<TemplateEditor />} />
        <Route path="template-edit/:id" element={<TemplateEditor />} />
        <Route path="templates" element={<Templates />} />
        <Route path="about" element={<About />} />
        <Route path="contacts" element={<Contacts />} />
      </Route>
    </Routes>
  );
};
```

Сторінка «Home.jsx»

```jsx
import React from 'react';
import {
  Button,
  Content,
  Description,
  Highlight,
  HomeContainer,
  MainContent,
  Subtitle,
  Title,
} from './Home.styled'; ← #2-11 import
import { useDispatch, useSelector } from 'react-redux';
import { setShowAuthForm } from 'redux/auth/authSlice';
import { selectUser } from 'redux/selectors';
import LogoImg from '../../images/Aidocs.png';    "Aidocs": Unknown word.

const HomePage = () => {
  const user = useSelector(selectUser);
  const dispatch = useDispatch();
  const handleToggleAuthForm = () => {
    dispatch(setShowAuthForm(true));
  };
  return (
    <Content>
      <MainContent>
        <img src={LogoImg} alt="logo" width={'450px'} height={'400px'} />
        <HomeContainer>
          <Title>Ласкаво просимо до AI DOCS!</Title>    "Ласкаво": Unknown word.
          <Subtitle>
            Інтелектуальна система створення шаблонів документів підприємця.    "Інтелектуальна": Unknown word.
          </Subtitle>
          <Description>
            Знаходьте і створюйте різноманітні шаблони документів для вашого    "Знаходьте": Unknown word.
            бізнесу з легкістю за допомогою нашої інтелектуальної системи AI    "бізнесу": Unknown word.
            DOCS. Наша платформа не лише дозволяє ефективно створювати нові    "Наша": Unknown word.
            шаблони, але й надає можливість шукати та використовувати існуючі у    "шаблони": Unknown word.
            нашій обширній базі документів. Забезпечте собі зручність, безпеку    "нашій": Unknown word.
            та швидкість в управлінні вашою документацією.    "швидкість": Unknown word.
          </Description>
          <Highlight>Зручно. Безпечно. Швидко.</Highlight>    "Зручно": Unknown word.
          {!user && (
            <Button onClick={handleToggleAuthForm}>Створити шаблон</Button>    "Створити": Unknown word.
          )}
        </HomeContainer>
      </MainContent>
    </Content>
  ); ← #23-47 return
}; ← #17-48 const HomePage = () =>

export default HomePage;
```

Компонент «Home.styled.js»

src > pages > Home > Home.styled.js > ...

```
import styled from 'styled-components';
import FonImage from '../../images/Home.png';

export const HomeContainer = styled.div`
  text-align: center;
  max-width: 80%;
  max-height: 100%;
  padding: 20px;
  background-color: rgba(255, 255, 255, 0.8);
`;

export const Title = styled.h1`
  font-size: 30px;
  font-weight: bold;
  margin-bottom: 10px;
`;

export const Subtitle = styled.p`
  font-size: 24px;
  margin-bottom: 10px;
`;

export const Description = styled.p`
  margin-bottom: 20px;
  font-size: 20px;
`;
export const Content = styled.div`
  height: 98vh;
  max-width: calc(100% - 300px);
  flex-grow: 1;
  background-image: url(${FonImage});
  background-repeat: no-repeat;
  background-size: cover;
  background-position: 50% 50%;
`;
export const Highlight = styled.p`
  font-size: 16px;
  font-weight: bold;
  color: #106701;
`;
export const Button = styled.button`
  padding: 10px 20px;
  font-size: 16px;
  color: #fff;
  background-color: #007bff;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  &:hover {
    background-color: yellow;
    color: #000;
  }
`;
export const MainContent = styled.div`
  width: 100%;
  height: 100%;
  display: flex;
  gap: 30px;
  justify-content: center;
  align-items: center;
  flex-direction: column;
`;

```

Сторінка «TemplateEditor.jsx»

```
import React, { useState } from 'react';
import {
  Content,
  MainContent,
  Form,
  Button,
  EditorContainer,
  Input,
  H2,
  Label,
  InputName,
  PdfBox,
} from './TemplateEditor.styled'; ← #2-13 import

import html2pdf from 'html2pdf.js';
import ConvertApi from 'convertapi-js';    "convertapi": Unknown word.
import MyEditor from 'components/MyEditor/MyEditor';
import { useDispatch, useSelector } from 'react-redux';
import { selectTemplates, selectUser } from 'redux/selectors';
import {
  addTemplate,
  setImage,
  updateTemplate,
} from 'redux/templates/operationsTemplate'; ← #20-24 import
import { useParams } from 'react-router-dom';

const TemplateEditor = () => {
  const user = useSelector(selectUser);
  const getTemplates = useSelector(selectTemplates);
  const { id } = useParams();
  const template = getTemplates.find(temp => temp.id === id);
  const dispatch = useDispatch();
  const [editorValue, setEditorValue] = useState(
    template ? template.description : ''
  );
  const [templateName, setTemplateName] = useState(
    template ? template.name : ''
  );
  const [templateImageUrl, setTemplateImageUrl] = useState(
    template ? template.templateImageUrl : ''
  );

  const convertApi = ConvertApi.auth('ce0Ute8WpA1WGppR');

  const handleChangeName = e => {
    setTemplateName(e.target.value);
  };

  const handleChange = (content, editor) => {
    setEditorValue(content);
  };
  const handleFileChange = async event => {
    const file = event.target.files[0];
    setTemplateName(file.name);
    const params = convertApi.createParams();
    params.add('file', file);

    try {
```

```
    try {
      const result = await convertApi.convert('docx', 'html', params);
      const result1 = await convertApi.convert('docx', 'jpg', params);
      // Отримайте посилання на завантаження HTML-файлу    "Отримайте": Unknown
      const htmlFileUrl = result.files[0].Url;
      const htmlFileUrl1 = result1.files[0].Url;

      const prevUrl = await setImage(
        'previous',
        htmlFileUrl1,
        result1.files[0].FileId
      ); ← #65-69 const prevUrl = await setImage
      // Завантажте вміст HTML-файлу    "Завантажте": Unknown word.
      setTemplateImageUrl(prevUrl);
      const response = await fetch(htmlFileUrl);
      const content = await response.text();

      setEditorValue(content);
    } catch (error) {
      console.error('Error converting Word file:', error);
    }
  }; ← #52-79 const handleFileChange = async event ⇒

  const saveTemplate = () ⇒ {
    dispatch(
      id
        ? updateTemplate({
            id: template.id,
            userEmail: user.email,
            name: templateName,
            description: editorValue,
            templateImageUrl: templateImageUrl,
          }) ← #84-90 ? updateTemplate
        : addTemplate({
            userEmail: user.email,
            name: templateName,
            description: editorValue,
            templateImageUrl: templateImageUrl,
          }) ← #91-96 : addTemplate
    ); ← #82-97 dispatch
  }; ← #81-98 const saveTemplate = () ⇒

  const handleGeneratePDF = () ⇒ {
    const content = editorValue;

    html2pdf(content, {
      margin: 10,
      filename: 'document.pdf',
      image: { type: 'jpeg', quality: 0.98 },
      html2canvas: { scale: 2 },
      jsPDF: { unit: 'mm', format: 'a4', orientation: 'portrait' },
    }); ← #103-109 html2pdf
  }; ← #100-110 const handleGeneratePDF = () ⇒

  return (
    <Content edit={template?.id}>
      <MainContent>
        <H2>{template?.id ? 'Редагування шаблону' : 'Створення шаблону'}</H2>
```

```
<H2>{template?.id ? 'Редагування шаблону' : 'Створення шаблону'}</H2>    "Редагування": U
<Form>
  <div
    style={{
      marginBottom: '10px',
      width: '100%',
      display: 'flex',
      gap: '10px',
      alignItems: 'center',
    }} ← #118-124 style=
  >
    <Label>Ім'я шаблону</Label>    "Ім'я": Unknown word.
    <InputName
      type="text"
      placeholder="Ім'я шаблону"    "Ім'я": Unknown word.
      value={templateName}
      onChange={handleChangeName}
    />
    {!user?.email ? (
      <div
        style={{
          backgroundColor: '#d23c46',
          color: 'white',
          display: 'flex',
          justifyContent: 'center',
          alignItems: 'center',
          boxSizing: 'border-box',
          borderRadius: '5px',
        }} ← #135-143 style=
      >
        <span
          style={{
            margin: '0 auto',
            display: 'block',
            textAlign: 'center',
          }} ← #146-150 style=
        >
          Для можливості редагування зареєструйтесь    "можливості": Unknown word.
        </span>
      </div>
    ) : (
      <>
        <Button onClick={saveTemplate}>Зберегти шаблон</Button>    "Зберегти": Unknown wo
        <Button onClick={handleGeneratePDF}>Завантажити PDF</Button>    "Завантажити": Ur
      </>
    )} ← #133-160 />
  </div>
  <EditorContainer>
    {!user?.email ? (
      <PdfBox url={template?.templateImageUrl} />
    ) : (
      <MyEditor
        editorValue={editorValue}
        apiKey="a85ckh1rvs1kxnitvo54z16evw4mvc00gqs3ukonw5rzvgd1"
        init={{
          height: '100%',
          menubar: true,
```

```
                branding: false,
                plugins: [
                  'advlist autolink lists link image charmap print preview anchor',    "advlist": Unknown word.
                  'searchreplace visualblocks code fullscreen',    "searchreplace": Unknown word.
                  'insertdatetime media table paste code help wordcount',    "insertdatetime": Unknown word.
                ], ← #173-177 plugins:
                toolbar:
                  'undo redo | formatselect | ' +    "formatselect": Unknown word.
                  'bold italic backcolor | alignleft aligncenter ' +    "backcolor": Unknown word.
                  'alignright alignjustify | bullist numlist outdent indent | ' +    "alignright": Unknown word.
                  'removeformat | help',    "removeformat": Unknown word.
                // Додаткові опції    "Додаткові": Unknown word.
                fontsize_formats: '8pt 10pt 12pt 14pt 18pt 24pt 36pt',
                language: 'uk',
                content_style: 'body { font-family: Arial, sans-serif; }',
                automatic_uploads: true,
                file_picker_types: 'image',
                file_picker_callback: (callback, value, meta) ⇒ {
                  // Додайте свій власний код для вибору файлів    "Додайте": Unknown word.
                },
                // Інші опції за потреби    "Інші": Unknown word.
              }} ← #169-193 init=
              handleChange={handleChange}
            />
          )} ← #163-196 <EditorContainer>
        </EditorContainer>
        {template?.id ? null : (
          <Input type="file" onChange={handleFileChange} />
        )}
      </Form>
    </MainContent>
  </Content>
); ← #112-204 return
}; ← #27-205 const TemplateEditor = () ⇒

export default TemplateEditor;
```

Сторінка «TemplateEditor.styled.jsx»

```jsx
import styled from 'styled-components';
import FonImage from '../../images/blue.jpg';
import FonImage1 from '../../images/orange.jpg';
import SiImage from '../../images/aidocs1.png';     "aidocs": Unknown word.
export const Content = styled.div`
  height: 98vh;
  max-width: calc(100% - 300px);
  flex-grow: 1;
  background-image: url(${props => (props.edit ? FonImage1 : FonImage)});
  background-repeat: no-repeat;
  background-size: cover;
  background-position: 50% 50%;
  overflow: hidden;
`;
export const MainContent = styled.div`
  display: flex;
  width: 100%;
  height: 100%;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  flex: 1;
  overflow-y: auto;
`;

export const Form = styled.div`
  background-color: rgba(255, 255, 255, 1);
  padding: 10px;
  border-radius: 8px;
  height: 80%;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  width: 90%;
  display: flex;
  flex-direction: column;
`;
```

```
export const EditorContainer = styled.div`
  width: 100%;
  flex-grow: 1;
`;
export const PdfBox = styled.div`
  margin: 0 auto;
  width: 50%;
  height: 100%;
  background-image: url(${SiImage}), url(${props => props.url});
  background-position: center, center;
  background-size: 50px, contain;
  background-repeat: space, no-repeat;
`;

export const Button = styled.button`
  padding: 10px 20px;
  font-size: 16px;
  color: #fff;
  background-color: #007bff;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  margin-top: 10px;
  &:hover {
    background-color: #0056b3;
  }
`;
export const Input = styled.input`
  padding: 10px;
  font-size: 16px;
  color: #333; /* Колір тексту */    "Колір": Unknown word.
  background-color: #fff; /* Колір фону */    "Колір": Unknown word.
  border: 1px solid #ccc; /* Тонка рамка */    "Тонка": Unknown word.
  border-radius: 5px; /* Закруглені кути */    "Закруглені": Unknown word.
  outline: none; /* Забирає контур при фокусі */    "Забирає": Unknown word.
  margin-top: 10px;
  width: 98%;

  &:hover,
  &:focus {
    border-color: #007bff; /* Колір рамки при наведенні чи фокусі */    "Колір": Unknown word.
  }
`;
export const InputName = styled.input`
  padding: 10px;
  font-size: 16px;
  color: #333; /* Колір тексту */    "Колір": Unknown word.
  background-color: #fff; /* Колір фону */    "Колір": Unknown word.
  border: 1px solid #ccc; /* Тонка рамка */    "Тонка": Unknown word.
  border-radius: 5px; /* Закруглені кути */    "Закруглені": Unknown word.
  outline: none; /* Забирає контур при фокусі */    "Забирає": Unknown word.
  margin-top: 10px;
  width: 55%;

  &:hover,
  &:focus {
    border-color: #007bff; /* Колір рамки при наведенні чи фокусі */    "рамки": Unknown word.
```

```
export const Label = styled.label`
  padding: 10px;
  font-size: 18px;
  font-weight: bold;
  color: #333; /* Колір тексту */    "тексту": Unknown word.
  margin-top: 10px;
`;
export const H2 = styled.h2`
  font-size: 40px;
  text-shadow: 2px 2px 4px #fff; /* Додає білу обводку */    "Додає": Unknown word.
`;
```

Сторінка «Templates.jsx»

```jsx
import React, { useEffect, useState } from 'react';
import { Content, MainContent } from './Templates.styled';
import TemplatesList from '../../components/TemplatesList/TemplatesList';
import { useDispatch, useSelector } from 'react-redux';

import { selectFavTemplates, selectTemplates } from 'redux/selectors';
import { H2 } from 'pages/TemplateEditor/TemplateEditor.styled';
import { fetchTemplates } from 'redux/templates/operationsTemplate';
import TemplateFind from '../../components/TemplateFind/TemplateFind';

const Templates = () => {
  const templates = useSelector(selectTemplates);
  const favTemplates = useSelector(selectFavTemplates);
  const [search, setSearch] = useState('');
  const [favorite, setFavorite] = useState('');
  const [allTemplates, setAllTemplates] = useState(
    templates?.length === 0 ? [] : templates
  );

  const dispatch = useDispatch();

  useEffect(() => {
    const getTemplates = async () => {
      try {
        const newTemplates = await dispatch(fetchTemplates());

        setAllTemplates(newTemplates.payload.templates);
      } catch (error) {
        console.error('Error fetching templates:', error);
      }
    }; ← #23-31 const getTemplates = async () =>

    getTemplates();
  }, [dispatch]); ← #22-34 useEffect
  useEffect(() => {
    if (search) {
      const searchTemplates = allTemplates.filter(temp =>
        temp.description.includes(search)
      );
      setAllTemplates(searchTemplates);  Warning: Maximum update depth exceeded. This can happen when a component
    } else setAllTemplates(templates);  Warning: Maximum update depth exceeded. This can happen when a component c
  }, [search, allTemplates, templates]); ← #35-42 useEffect
  useEffect(() => {
    if (favorite) {
      const favoriteTemplates = allTemplates.filter(temp =>
        favTemplates.some(obj => obj.uid === temp.id)
      );

      setAllTemplates(favoriteTemplates);
    } else setAllTemplates(templates); ← #44-50 if (favorite)
  }, [favorite, allTemplates, favTemplates, templates]); ← #43-51 useEffect
  return (
    <Content search={search + favorite}>
      <>
        <H2>Шаблони документів</H2>    "Шаблони": Unknown word.
        <TemplateFind onSearch={setSearch} onFavorite={setFavorite} />
        <MainContent>
          {allTemplates && <TemplatesList allTemplates={allTemplates} />}
        </MainContent>
      </>
    </Content>
  ); ← #52-62 return
}; ← #11-63 const Templates = () =>

export default Templates;
```

Компонент «Templates.styled.js»

```
import styled from 'styled-components';
import FonImage from '../../images/green.jpg';
import FonImage1 from '../../images/gray.jpg';
export const Content = styled.div`
  display: flex;
  flex-direction: column;
  align-items: center;
  height: 98vh;
  max-width: calc(100% - 300px);
  flex-grow: 1;
  background-image: url(${props => (props.search ? FonImage1 : FonImage)});
  background-repeat: no-repeat;
  background-size: cover;
  background-position: 50% 50%;
  overflow: hidden;
`;

export const MainContent = styled.div`
  background-color: rgba(255, 255, 255, 0.8);
  width: 90%;
  height: 73vh;
  box-sizing: border-box;
  padding: 10px 10px;
  border-radius: 8px;
  display: flex;
  justify-content: center;
  align-items: center;
  overflow: auto;
`;
export const Title = styled.h2`
  text-align: center;
`;
```

Сторінка «Contacts.jsx»

```
import React from 'react';
import {
  ContactContainer,
  ContactInfo,
  Content,
  Label,
  Link,
  MainContent,
  Subtitle,
  Title,
} from './Contacts.styled';
import LogoImg from '../../images/Aidocs.png';     "Aidocs": Unknown word.

const Contacts = () => {
  return (
    <Content>
      <MainContent>
        {' '}
        <img src={LogoImg} alt="logo" width={'450px'} height={'400px'} />
        <ContactContainer>
          <Title>Контакти</Title>     "Контакти": Unknown word.
          <Subtitle>
            Зв'яжіться з нами для отримання додаткової інформації:     "Зв'яжіться":
          </Subtitle>
          <ContactInfo>
            <Label>Email:</Label>{' '}
            <Link href="mailto:info@aidocs.com">info@aidocs.com</Link>
          </ContactInfo>
          <ContactInfo>
            <Label>Телефон:</Label>{' '}     "Телефон": Unknown word.
            <Link href="tel:+380980000000">+38 098 000 000 0</Link>
          </ContactInfo>
          <ContactInfo>
            <Label>Адреса:</Label> AI DOCS, вул. Центральна 10, м. Kmdsd,     "Адрес
            Україна     "Україна": Unknown word.
          </ContactInfo>
        </ContactContainer>
      </MainContent>
    </Content>
  );
};

export default Contacts;
```

Компонент «Contacts.styled.js»

```
import styled from 'styled-components';
import FonImage from '../../images/Contact.jpg';

export const ContactContainer = styled.div`
  max-width: 80%;
  max-height: 80%;
  padding: 20px;
  background-color: rgba(255, 255, 255, 0.8);
`;

export const Title = styled.h1`
  font-size: 24px;
  font-weight: bold;
  margin-bottom: 10px;
`;

export const Subtitle = styled.p`
  font-size: 18px;
  margin-bottom: 10px;
`;

export const ContactInfo = styled.div`
  margin-bottom: 10px;
`;

export const Label = styled.span`
  font-weight: bold;
`;

export const Link = styled.a`
  color: #007bff;
`;
export const Content = styled.div`
  height: 98vh;
  max-width: calc(100% - 300px);
  flex-grow: 1;
  background-image: url(${FonImage});
  background-repeat: no-repeat;
  background-size: cover;
  background-position: 50% 50%;
`;
export const MainContent = styled.div`
  width: 100%;
  height: 100%;
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: row-reverse;
`;
```

Сторінка «About.jsx»

```jsx
import React from 'react';
import {
  AboutContainer,
  Content,
  Description,
  MainContent,
  Paragraph,
  Subtitle,
  Title,
} from './About.styled.';
import LogoImg from '../../images/Aidocs.png';

const About = () => {
  return (
    <Content>
      <MainContent>
        <img src={LogoImg} alt="logo" width={'450px'} height={'400px'} />
        <AboutContainer>
          <Title>Про нас</Title>
          <Description>
            Інтелектуальна система створення шаблонів документів для підприємців
            AI DOCS забезпечує швидкі та надійні вантажні перевезення. Ми
            спеціалізуємося на наданні професійних послуг, сприяючи зв'язку між
            перевізниками та клієнтами.
          </Description>
          <Subtitle>Наша місія</Subtitle>
          <Paragraph>
            Наша місія полягає в забезпеченні наших клієнтів найкращими
            вантажними перевезеннями та високоякісним обслуговуванням. Ми
            прагнемо створити безпечний та надійний зв'язок за доступними
            цінами.
          </Paragraph>
          <Subtitle>Наші цінності</Subtitle>
          <Paragraph>
            - Професіоналізм: Ми співпрацюємо з досвідченими та надійними
            водіями, щоб гарантувати високу якість перевезень.
          </Paragraph>
          <Paragraph>
            - Відповідальність: Ми несемо відповідальність за кожне перевезення
            та гарантуємо його безпечну та своєчасну доставку.
          </Paragraph>
          <Paragraph>
            - Клієнтоорієнтованість: Ми завжди ставимо потреби та задоволення
            наших клієнтів на перше місце.
          </Paragraph>
        </AboutContainer>
      </MainContent>
    </Content>
  );
};

export default About;
```

Сторінка «About.styled.js»

pages > About > About.styled.js > MainContent

```
import styled from 'styled-components';
import FonImage from '../../images/About.jpg';

export const AboutContainer = styled.div`
  max-width: 80%;
  max-height: 80%;
  padding: 20px;
  background-color: rgba(255, 255, 255, 0.8);
`;

export const Title = styled.h1`
  font-size: 24px;
  font-weight: bold;
  margin-bottom: 10px;
`;

export const Description = styled.p`
  margin-bottom: 20px;
`;
export const Subtitle = styled.h2`
  font-size: 20px;
  font-weight: bold;
  margin-bottom: 10px;
`;

export const Paragraph = styled.p`
  margin-bottom: 10px;
`;
export const Content = styled.div`
  height: 98vh;
  max-width: calc(100% - 300px);
  flex-grow: 1;
  background-image: url(${FonImage});
  background-repeat: no-repeat;
  background-size: cover;
  background-position: 50% 50%;
`;
export const MainContent = styled.div`
  width: 100%;
  height: 100%;
  display: flex;
  justify-content: center;
  align-items: center;
`;

```

Компонент «TemplatesList.styled.js»

```
import { Link } from 'react-router-dom';
import styled from 'styled-components';

export const TransportationItem = styled.div`
  display: flex;
  width: 100%;
  justify-content: space-between;
  border-radius: 4px;
  background-color: rgba(255, 255, 255, 0.8);
  min-height: 100px;
  padding: 5px;
  font-weight: 700;
  box-shadow: 0 6px 10px rgba(0, 0, 0, 0.1);
  align-items: center;
  padding-left: 20px;
  margin: 10px 0;
  @media (max-width: 768px) {
    /* Застосовуємо стилі, коли ширина екрану менше або рівна 768px */    "Застосовуємо": Unknown
    flex-direction: column;
    align-items: stretch;
    padding: 10px;
  }
`;

export const EditButton = styled(Link)`
  background-color: #3498db; /* Колір для кнопки редагування */    "Колір": Unknown word.
  color: #ffffff;
  border: none;
  display: flex;
  justify-content: center;
  align-items: center;
  border-radius: 5px;
  cursor: pointer;
  transition: background-color 0.3s;
  width: 30px;
  height: 30px;
  &:hover {
    background-color: #2980b9; /* Змінений колір при наведенні */    "Змінений": Unknown word.
  }
`;

export const DeleteButton = styled.button`
  background-color: #e74c3c; /* Колір для кнопки видалення */    "Колір": Unknown word.
  color: #ffffff;
  border: none;
  width: 30px;
  height: 30px;
  padding: 5px;
  border-radius: 5px;
  cursor: pointer;
  transition: background-color 0.3s;
```

```
  &:hover {
    background-color: #c0392b; /* Змінений колір при наведенні */    "Змінений": Unknown w
  }
`;
export const FavButton = styled.button`
  background-color: ${props =>
    props.col === -1 ? '#808080' : '#124212'}; /* Колір для кнопки видалення */    "Колір"
  color: #ffffff;
  border: none;
  width: 30px;
  height: 30px;
  padding: 5px;
  border-radius: 5px;
  cursor: pointer;
  transition: background-color 0.3s;

  &:hover {
    background-color: ${props =>
      props.col === -1
        ? '#333333'
        : '#10392b'}; /* Змінений колір при наведенні */    "Змінений": Unknown word.
  }
`;
export const Rout = styled.div`
  width: 10%;
  margin: 2px;
  border-radius: 4px;
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: column;
  gap: 5px;
  color: black;
  div {
    font-size: 10px;
    word-break: break-all;
    text-align: center;
    padding: 2px;
  }

  @media (max-width: 768px) {
    /* Застосовуємо стилі, коли ширина екрану менше або рівна 768px */    "Застосовуємо": 
    width: 100%;
  }
`;
export const RoutH = styled.div`
  width: 35%;
  height: 100%;
  margin-right: 10px;
  border-radius: 4px;
  display: flex;
  justify-content: flex-end;
  align-items: center;

  gap: 20px;
  color: black;
  div {
    font-size: 10px;
    word-break: break-all;
    text-align: center;
    padding: 2px;
  }

  @media (max-width: 768px) {
    /* Застосовуємо стилі, коли ширина екрану менше або рівна 768px */    "Застосовуємо": 
    width: 100%;
  }
`;
```

```
,
export const Container = styled.div`
  height: 100%;
  width: 90%;
  display: flex;
  flex-direction: column;
`;

export const P = styled.p`
  font-size: 24px;
  width: 100%;
`;
export const Image = styled.img`
  height: 100%;
  width: 110px;
  border: 1px solid rgba(0, 0, 0, 0.7);
  border-radius: 5px;
  box-shadow: 0 6px 10px rgba(0, 0, 0, 0.1);
  &:hover {
    scale: 1.2;
  }
  @media (max-width: 768px) {
    /* Застосовуємо стилі, коли ширина екрану менше або рівна 768px */    "Застосовуємо": Unknown
    height: 40%;
    width: 40%;
  }
`;

```

Компонент «TemplateFind.jsx»

```
import React, { useState } from 'react';
import { Button, FindContainer, Input } from './TemplateFind.styled';

const TemplateFind = ({ onSearch, onFavorite }) => {
  const [searchTerm, setSearchTerm] = useState('');

  const handleInputChange = event => {
    setSearchTerm(event.target.value);
  };

  const handleSearchClick = () => {
    onSearch(searchTerm);
  };
  const handleFavoriteClick = () => {
    onFavorite(prev => (prev === 'true' ? '' : 'true'));
  };
  return (
    <FindContainer>
      <Input
        type="text"
        placeholder="Введіть термін пошуку"     "Введіть": Unknown word.
        value={searchTerm}
        onChange={handleInputChange}
      />
      <Button onClick={handleSearchClick}>Пошук</Button>     "Пошук": Unknown wo
      <Button onClick={handleFavoriteClick}>Обрані</Button>     "Обрані": Unknow
    </FindContainer>
  ); ← #17-28 return
}; ← #4-29 const TemplateFind = ({ onSearch, onFavorite }) =>

export default TemplateFind;

```

Компонент «Layout.jsx»

```jsx
import { Link, Outlet } from 'react-router-dom';
import { ToastContainer } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import {
  Container,
  Header,
  ResponsiveLayout,
  Sidebar,
  Navigation,
  NavigationLink,
  AuthFormContainer,
} from './Layout.styled';
import LogoImg from '../../images/Aidocs.png';
import AuthForm from 'components/AuthForm/AuthForm';
import { useSelector } from 'react-redux';

const Layout = () => {
  const user = useSelector(state => state.auth.user);
  return (
    <ResponsiveLayout>
      <Container>
        <Outlet />
        <Sidebar>
          <Header>
            <Link to="/">
              <img src={LogoImg} alt="logo" width={'150px'} height={'120px'} />
            </Link>
          </Header>

          {!user ? (
            <Navigation>
              <NavigationLink to="/">Головна</NavigationLink>
              <NavigationLink to="/templates">Шаблони</NavigationLink>
              <NavigationLink to="/about">Про нас</NavigationLink>
              <NavigationLink to="/contacts">Контакти</NavigationLink>
            </Navigation>
          ) : (
            <Navigation>
              <NavigationLink to="/">Головна</NavigationLink>
              <NavigationLink to="/templates">Шаблони</NavigationLink>
              <NavigationLink to="/template-edit">
                Створити шаблон
              </NavigationLink>
              <NavigationLink to="/about">Про нас</NavigationLink>
              <NavigationLink to="/contacts">Контакти</NavigationLink>
            </Navigation>
          )}
          <ToastContainer />
          <AuthFormContainer>
            <AuthForm />
          </AuthFormContainer>
        </Sidebar>
      </Container>
    </ResponsiveLayout>
  );
};
```

Компонент «Layout.styled.js»

```
import styled from 'styled-components';
import SideImage from '../../images/fon.jpg';
import { NavLink } from 'react-router-dom';

const Container = styled.div`
  display: flex;
  flex-wrap: no-wrap;
`;

const Sidebar = styled.div`
  width: 300px;
  background-image: url(${SideImage});
  background-repeat: no-repeat;
  background-size: cover;
  background-position: 0% 50%;

  height: 98vh;
  display: flex;
  flex-direction: column;

  box-shadow: 0px 0px 25px 0px rgba(0, 0, 0, 0.7);
`;

const Header = styled.div`
  padding: 15px;
  display: flex;
  justify-content: center;
`;

const ResponsiveLayout = styled.div`
  @media (max-width: 468px) {
    ${Container} {
      flex-direction: column;
    }
    ${Sidebar} {
      width: 100%;
    }
  }
`;

export const NavigationLink = styled(NavLink)`
  margin-bottom: 15px;
  display: block;
  padding: 8px;
  text-align: center;
  color: #fff;
  background-color: #007bff;
  border-radius: 6px;
  text-decoration: none;
  transition: all 0.3s ease;
  &.active {
    background-color: yellow;
    color: #000;
  }
  &:hover {
    background-color: yellow;
    color: #000;
  }
`;
export const AuthFormContainer = styled.div`
  margin-top: auto;
  padding: 16px;
  margin-bottom: 50px;
`;

export { ResponsiveLayout, Container, Sidebar, Header };
```

Компонент « MyEditor.jsx»

```jsx
import React from 'react';
import { Editor } from '@tinymce/tinymce-react';    "tinymce": Unknown word.

const MyEditor = ({ editorValue, handleChange }) => {
  return (
    <Editor
      initialValue={editorValue}
      apiKey="a85ckh1rvs1kxnitvo54z16evw4mvc00gqs3ukonw5rzvgd1"
      init={{
        height: '100%',
        menubar: true,

        branding: false,
        plugins: [
          'advlist autolink lists link image charmap print preview anchor',
          'searchreplace visualblocks code fullscreen',    "searchreplace": Unk
          'insertdatetime media table paste code help wordcount',    "insertdat
        ], ← #14-18 plugins:
        toolbar:
          'undo redo | formatselect | ' +    "formatselect": Unknown word.
          'bold italic backcolor | alignleft aligncenter ' +    "backcolor": Un
          'alignright alignjustify | bullist numlist outdent indent | ' +    "a
          'removeformat | help',    "removeformat": Unknown word.
        // Додаткові опції    "Додаткові": Unknown word.
        fontsize_formats: '8pt 10pt 12pt 14pt 18pt 24pt 36pt',
        language: 'uk',
        content_style: 'body { font-family: Arial, sans-serif; }',
        automatic_uploads: true,
        file_picker_types: 'image',
        file_picker_callback: (callback, value, meta) => {
          // Додайте свій власний код для вибору файлів    "Додайте": Unknown w
        },
        // Інші опції за потреби    "Інші": Unknown word.
      }} ← #9-34 init=
      onEditorChange={handleChange}
    />
  ); ← #5-37 return
}; ← #4-38 const MyEditor = ({ editorValue, handleChange }) =>

export default MyEditor;
```

Компонент «AuthForm.styled.js»

```
import styled from 'styled-components';

export const AuthFormErrorMessage = styled.p`
  color: red;
  margin-bottom: 16px;
`;

export const AuthFormSuccessMessage = styled.p`
  color: green;
  margin-bottom: 16px;
`;

export const AuthFormContainer = styled.div`
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
`;

export const AuthFormInput = styled.input`
  width: 90%;
  height: 20px;
  padding: 8px;
  margin-bottom: 16px;
  border: 1px solid #ccc;
  border-radius: 4px;
`;
export const AuthFormButtonContainer = styled.div`
  display: flex;
  justify-content: center;
  align-items: center;
`;
export const AuthFormButton = styled.button`
  margin-right: 20px;
  padding-top: 5px;
  width: 40px;
  height: 40px;
  background-color: #007bff;
  color: #fff;
  border: none;
  border-radius: 10px;
  font-size: 16px;
  cursor: pointer;
  &:hover {
    background-color: yellow;
    color: #000;
  }
`;
```

Компонент «opertionsActions.js»

```
import { createAsyncThunk } from '@reduxjs/toolkit';    "reduxjs": Unknown word.
import { toast } from 'react-toastify';    "toastify": Unknown word.
import {
  GoogleAuthProvider,
  createUserWithEmailAndPassword,
  onAuthStateChanged,
  signInWithEmailAndPassword,
  signInWithPopup,
  signOut,
} from 'firebase/auth';
// import { setError, setLoading, setUser } from './authSlice';
import { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';

const firebaseConfig = {
  apiKey: 'AIzaSyAXxd5BLFD3cjyRh4yRnr-CKRAsYniXUek',    "BLFD": Unknown word.
  authDomain: 'ai-docs-1807d.firebaseapp.com',    "firebaseapp": Unknown word.
  projectId: 'ai-docs-1807d',
  storageBucket: 'ai-docs-1807d.appspot.com',    "appspot": Unknown word.
  messagingSenderId: '722774303553',
  appId: '1:722774303553:web:8f91361d17034c050bf7c0',
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);

// Initialize Firebase Authentication and get a reference to the service
const auth1 = getAuth(app);

// Створюємо провайдера Google для авторизації через Google    "Створюємо": Unknown word.
const googleProvider = new GoogleAuthProvider();

export const register = createAsyncThunk(
  'auth/register',
  async ({ email, password }, { rejectWithValue }) => {
    try {
      const userCredential = await createUserWithEmailAndPassword(
        auth1,
        email,
        password
      );
      const user = userCredential.user;
      return user;
    } catch (e) {
      toast.error(`User with email ${email} is already registered!`);

      return rejectWithValue(e.message);
    }
  }
);
export const login = createAsyncThunk(
  'auth/login',
  async ({ email, password }, { rejectWithValue }) => {
    try {
      const userCredential = await signInWithEmailAndPassword(
        auth1,
        email,
        password
      );
      const user = userCredential.user;
      return user;
    } catch (e) {
      toast.error(`User ${email} is login error.`);

      return rejectWithValue(e.message);
    }
  }
);
export const loginWithGoogle = createAsyncThunk(
  'auth/loginWithGoogle',
```

Компонент « helpFunctionAuth.js»

```
export const onPending = state => {
  state.isRefreshing = true;
};

export const onRejected = state => {
  state.isRefreshing = false;
};

export const onFulfilledRegisterLogin = (state, { payload }) => {
  state.user = { displayName: payload.displayName, email: payload.email };
  state.isLogged = true;
};

export const onFulfilledLogOut = state => {
  state.user = null;
  state.isLogged = false;
};
export const onFulfilledRefresh = (state, { payload }) => {
  state.isLogged = payload;
  state.isRefreshing = false;
};
```

Компонент «authSlice.js»

```
import { createSlice } from '@reduxjs/toolkit';    "reduxjs": Unknown word.
import {
  login,
  loginWithGoogle,
  logout,
  register,
  userRefresh,
} from './opertionsActions';    "opertions": Unknown word.
import {
  onFulfilledLogOut,
  onFulfilledRefresh,
  onFulfilledRegisterLogin,
  onPending,
  onRejected,
} from './helpFunctionAuth';

export const authSlice = createSlice({
  name: 'auth',
  initialState: {
    user: null,
    loading: false,
    error: null,
    showAuthForm: false,
  },
  reducers: {
    setShowAuthForm: (state, action) => {
      state.showAuthForm = action.payload;
    },
  },
  extraReducers: builder => {
    builder
      .addCase(register.fulfilled, onFulfilledRegisterLogin)
      .addCase(login.fulfilled, onFulfilledRegisterLogin)
      .addCase(loginWithGoogle.fulfilled, onFulfilledRegisterLogin)
      .addCase(logout.fulfilled, onFulfilledLogOut)
      .addCase(userRefresh.fulfilled, onFulfilledRefresh)
      .addCase(userRefresh.pending, onPending)
      .addCase(userRefresh.rejected, onRejected);
  },
});

export const { setShowAuthForm } = authSlice.actions;
```