МІНІСТЕРСТВО ВНУТРІШНІХ СПРАВ УКРАЇНИ

ЛЬВІВСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ВНУТРІШНІХ СПРАВ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ УПРАВЛІННЯ, ПСИХОЛОГІЇ ТА БЕЗПЕКИ

Кафедра інформаційних технологій

«РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СТВОРЕННЯ ШАБЛОНІВ ДОКУМЕНТІВ ДЛЯ ПІДРОЗДІЛІВ НАЦІОНАЛЬНОЇ ПОЛІЦІЇ»

кваліфікаційна робота
здобувача вищої освіти
4 курсу денної форми навчання
Діани ЛЕВЧЕНКО

Науковий керівник:
доцент, кандидат технічних наук
Ярко КУЛЕШНИК

Рецензент:
доцент, кандидат технічних наук
Світлана ЯЦИШИН

Кваліфікаційна робота допущена до захисту
«___» _____ 2026 р., протокол № ____
Завідувач кафедри інформаційних технологій
_____ **Олег ЗАЧЕК**
(підпис)

Львів
2026

АНОТАЦІЯ

ЛЕВЧЕНКО Д. Розроблення програмного забезпечення для створення шаблонів документів для підрозділів національної поліції. – Рукопис.

Дослідження на здобуття освітнього ступеня ″бакалавр″ за спеціальністю 126 ″Інформаційні системи та технології″. – Львівський державний університет внутрішніх справ, МВС України, Львів, 2026.

Кваліфікаційна робота присвячена проєктуванню та програмній реалізації Web-застосунку AiDocs, призначеного для оптимізації процесів генерування та управління цифровою документацією. Система пропонує розширений функціонал для оперативного створення документів, забезпечуючи високу продуктивність та ергономічність користувацького інтерфейсу.

Завдяки комбінації реактивного інтерфейсу на базі React та високої продуктивності середовища Node.js на бекенді, реалізована платформа характеризується стабільністю роботи та можливістю легкого розширення функціоналу в майбутньому.

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

ABSTRACT

LEVCHENKO D. Development of software for creating document templates for national police units. – Manuscript.

Research for the degree of ″Bachelor″ in specialty 126 ″Information Systems and Technologies″. – Lviv State University of Internal Affairs, Ministry of Internal Affairs of Ukraine, Lviv, 2026.

The qualification work is devoted to the design and software implementation of the AiDocs Web application, designed to optimize the processes of generating and managing digital documentation. The system offers advanced functionality for operational document creation, ensuring high productivity and ergonomics of the user interface.

Due to the combination of a reactive interface based on React and high productivity of the Node.js environment on the backend, the implemented platform is characterized by stability of operation and the possibility of easy expansion of functionality in the future.

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

ЗМІСТ

ВСТУП

У сучасних умовах цифровізації економіки системи електронного документообігу (СЕД) відіграють стратегічну роль у модернізації та оптимізації діловодних процесів як у державних структурах, так і в комерційних організаціях. Перехід до автоматизованого управління документацією сприяє стандартизації управлінської діяльності, забезпечує високу швидкість доступу до інформації та дозволяє мінімізувати вплив людського фактора під час виконання рутинних операцій. Використання інтелектуальних платформ, таких як AiDocs, дозволяє не лише впорядкувати потоки документації, а й забезпечити прозорість усіх етапів життєвого циклу документа — від створення та редагування до архівування. Такі системи критично важливі для покращення внутрішньої комунікації в організаціях, оскільки вони гарантують оперативний обмін даними між підрозділами та підвищують загальну продуктивність колективу в умовах динамічного бізнес-середовища.

Інноваційна система AiDocs, спроєктована на базі сучасного фреймворку React, пропонує розширений функціонал для роботи з корпоративним контентом. Ключовою перевагою платформи є можливість швидкої генерації нових документів на основі гнучких шаблонів, що суттєво скорочує часові витрати на підготовку типової звітності та договорів. Інтегрована база шаблонів дозволяє користувачам обирати оптимальні структурні рішення та адаптувати їх під специфічні потреби, а реалізація механізмів експорту у формат PDF та підготовки до друку забезпечує універсальність використання створених документів поза межами системи.

Програмний продукт повинен забезпечувати користувачам можливість оперативного проєктування нових документів на основі існуючих прототипів, реалізуючи механізми клонування та спадковості структурних елементів. У межах системи необхідно впровадити розширений інструментарій для персоналізації контенту, що передбачає інтеграцію варіативних полів,

статичних та динамічних блоків, а також специфічних елементів верстки для адаптації шаблонів під конкретні бізнес-задачі.

Система має підтримувати гнучке налаштування обраних макетів у режимі реального часу, забезпечуючи їх повну відповідність індивідуальним потребам та стандартам діловодства. Обов'язковою функціональною вимогою є реалізація модуля експорту, який дозволяє здійснювати рендеринг та завантаження сформованих матеріалів у загальноприйнятому форматі PDF для забезпечення подальшої мобільності та зручності друку. Особлива увага при проєктуванні надається принципам ергономічності та доступності інтерфейсу, що гарантує низький поріг входження та високу продуктивність роботи користувачів незалежно від їхньої технічної підготовки.

Актуальність роботи полягає не тільки у виконанні аналізу предметної області, а також у галузі проєктування та програмного реалізування Web-застосунку AiDocs, призначеного для оптимізації процесів генерування та управління цифровою документацією.

Об’єктом дослідження є процеси автоматизації документообігу та створення цифрових контент-платформ у сучасному вебредакторі.

Предметом дослідження є методологія та програмна реалізація основних підходів до розробки систем управління документацією з використанням бібліотеки React, середовища виконання Node.js та хмарних сервісів збереження даних.

Метою роботи є проєктування та розробка високоефективної системи керування документами AiDocs, яка забезпечує автоматизацію створення, редагування та зберігання документації з можливістю подальшого масштабування та інтеграції нових функціональних модулів.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати архітектурні особливості сучасних СЕД.
- розробити компонентну структуру інтерфейсу на базі React.
- реалізувати логіку обробки даних у середовищі Node.js.
- забезпечити механізм генерації та конвертації документів у формат

PDF.

Система має підтримувати гнучке налаштування обраних макетів у режимі реального часу, забезпечуючи їх повну відповідність індивідуальним потребам та стандартам діловодства. Обов'язковою функціональною вимогою є реалізація модуля експорту, який дозволяє здійснювати рендеринг та завантаження сформованих матеріалів у загальноприйнятому форматі PDF для забезпечення подальшої мобільності та зручності друку. Особлива увага при проєктуванні приділяється принципам ергономічності та доступності інтерфейсу, що гарантує низький поріг входження та високу продуктивність роботи користувачів незалежно від їхньої технічної підготовки.

Наукова новизна отриманих результатів полягає у розробці та впровадженні архітектури попередньо запрограмованого контролера управління станом, що дозволяє значно пришвидшити обробку запитів та ініціалізацію робочого середовища редактора, підвищуючи загальну швидкість реакції системи на дії користувача.

Практичне значення роботи полягає у створенні готового до експлуатації програмного продукту з високим рівнем юзабіліті. Система дозволяє користувачам без спеціальних навичок програмування чи верстки ефективно завантажувати, редагувати та використовувати професійні шаблони, забезпечуючи високу якість фінальної документації та знижуючи операційні витрати підприємства на діловодство.

Структура дипломної роботи охоплює вступ, три розділи, підсумкові висновки та перелік використаних джерел, 1 додаток.

У першому розділі здійснено дослідження та огляд предметної області, виконано аналіз можливостей та переваг проєктованого застосунку, програмних засобів. Другий розділ присвячено обгрунтуванню вибору інформаційного забезпечення. У третьому розділі розглянуто програмне та технічне забезпечення для реалізування програмної частини, налаштування технологій.

Загальний обсяг роботи становить 81 сторінки. У роботі подано 41

рисунок. Для підготовки дослідження використано 17 джерел. Наявний 1 додаток.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Огляд проблемної області.

Сучасні системи електронного документообігу (СЕД) є універсальним інструментом, що інтегрується в управлінські цикли як державних структур, так і приватних підприємств. Основне призначення таких систем полягає в уніфікації діловодних процесів, структуруванні інформаційних потоків та оптимізації діяльності персоналу. Завдяки впровадженню СЕД забезпечується безперешкодний доступ до документарного фонду, автоматизуються рутинні операції з моніторингу, пошуку релевантної інформації та генерації аналітичної звітності.

У контексті розбудови електронного урядування навантаження на інформаційні системи суттєво зростає, досягаючи показника у кілька тисяч запитів щоденно. Використання сучасних технологічних стеків, зокрема фреймворку React, дозволяє створювати гнучкі інтерфейси для швидкої розробки нових шаблонів на базі існуючих прототипів. Такий підхід не лише прискорює обробку кореспонденції, а й мінімізує ризик виникнення помилок, зумовлених людським фактором, готуючи при цьому необхідний масив даних для прийняття стратегічних рішень [1].

Історична трансформація методів управління документацією розпочалася у 1980-х роках, що було зумовлено масовою комп'ютеризацією. Впровадження серверних технологій та централізованих мейнфреймів дозволило організаціям перейти до електронного формату зберігання даних. Паралельно з цим, поява засобів сканування забезпечила конвертацію паперових носіїв у цифровий вигляд.

Цінність сучасних СЕД для бізнес-середовища визначається не лише великими обсягами пам'яті, а й складною архітектурою класифікації. Ефективність впорядкування документів досягається через використання метаданих — спеціальних атрибутів, що описують властивості об'єкта та

забезпечують точність його ідентифікації в системі.

Документообіг є фундаментом будь-якої організаційної діяльності, оскільки в ньому відображаються фінансові, управлінські та виробничі цикли. Пандемія COVID-19 стала каталізатором різкого збільшення обсягів цифрової кореспонденції, що, у свою чергу, підвищило навантаження на працівників служб діловодства. Наслідком цього стало зростання часових та ресурсних витрат на механічну обробку великих масивів даних[2].

За сучасних умов пріоритетним напрямом удосконалення СЕД є інтеграція методів машинного навчання та інтелектуального аналізу даних. Це дозволяє забезпечити автоматизацію повного життєвого циклу документа з мінімальною участю людини. Розроблені технологічні рішення базуються на:

- моделюванні процесів вилучення ключової інформації;
- використанні онтологій предметної області;
- застосуванні лінгвістичних знань, структурованих у вигляді фактографічних моделей та спеціалізованих словників.

Такий підхід трансформує традиційні системи в інтелектуальні середовища автоматичної обробки текстів, що суттєво підвищує загальну результативність функціонування установ [2].

1.2. Можливості й переваги застосунку "AiDocs ".

Програмний комплекс AiDocs постає як інноваційне рішення у сфері автоматизації життєвого циклу документації, поєднуючи в собі передові архітектурні підходи та інтелектуальні інструменти управління. Фундаментом системи є фреймворк React, застосування якого зумовлює високу реактивність інтерфейсу та забезпечує безперебійну взаємодію користувача з динамічними елементами платформи. Завдяки реалізації компонентного підходу розробникам вдалося досягти високої модульності системи, що дозволяє користувачам ефективно оперувати складними функціональними блоками, мінімізуючи часові витрати на освоєння інструментарію. Центральною ланкою функціоналу AiDocs є можливість предиктивної генерації нових

шаблонів на основі існуючих прототипів, що значно прискорює процес підготовки специфічної документації та підвищує загальну продуктивність праці в межах організації [3].

Важливою складовою екосистеми AiDocs є розгалужена база верифікованих шаблонів, яка дозволяє здійснювати прецизійний підбір необхідної структури документа з подальшою гнучкою кастомізацією під конкретні потреби суб'єкта. Кінцева обробка сформованих даних передбачає можливість виведення документів на друк або їх експорт у кросплатформний формат PDF, що гарантує збереження цілісності верстки та автентичності контенту. Водночас доступ до повного спектра операційних можливостей регламентується через систему обов'язкової реєстрації, яка виступає гарантом дотримання протоколів безпеки та конфіденційності персональної інформації.

Для суб'єктів, що перебувають на етапі ознайомлення з платформою, передбачено демонстраційний режим перегляду графічних копій шаблонів із накладеними захисними водяними знаками. Такий підхід дозволяє провести об'єктивну оцінку функціонального різноманіття та ергономічності програмного забезпечення без передчасного розкриття повних інтелектуальних ресурсів системи, забезпечуючи при цьому прозорість взаємодії між розробником та потенційним користувачем [4].

Висновки до розділу.

На основі проведеного аналізу особливостей та функціональних характеристик систем електронного документообігу (СЕД), зокрема інноваційного рішення AiDocs, можна сформулювати такі висновки:

Впровадження інтелектуальних систем управління документацією є об’єктивною вимогою сучасної парадигми цифровізації, що дозволяє оптимізувати діловодні процеси як у державному, так і в комерційному секторах. Застосування прогресивних архітектурних рішень, таких як фреймворк React, забезпечує високу операційну ефективність, що виражається у швидкій генерації шаблонів та мінімізації імовірності виникнення помилок, зумовлених людським фактором. Використання компонентного підходу до розробки інтерфейсів суттєво підвищує зручність роботи користувачів із великими масивами даних, перетворюючи механічну діяльність на структурований технологічний процес.

Еволюція СЕД від простих серверних сховищ до інтелектуальних платформ з розвиненою системою класифікації та метаданими засвідчує перехід до автоматизації повного життєвого циклу документа. Зокрема, такі рішення, як AiDocs, демонструють синергію між функціональною гнучкістю — можливістю швидкого налаштування документів та їх експорту в уніфіковані формати — та суворою політикою безпеки, яка передбачає розмежування прав доступу для авторизованих та ознайомчих користувачів. Такий підхід не лише гарантує цілісність інформації, а й сприяє прозорості взаємодії між системою та користувачем на етапах оцінювання її потенціалу.

Перспективність розвитку таких систем полягає у подальшій інтеграції методів машинного навчання, онтологічного моделювання предметних областей та лінгвістичного аналізу текстів. Це дозволить трансформувати традиційні інструменти реєстрації документації в автономні інтелектуальні системи, здатні до самостійного опрацювання великих обсягів інформації з мінімальним залученням людського ресурсу.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. React - відкрита JavaScript бібліотека.

Ефективність функціонування інтерфейсу системи AiDocs зумовлена інтеграцією бібліотеки **React**, ключовою перевагою якої є використання концепції віртуального DOM (**Virtual DOM**). Даний механізм дозволяє оптимізувати процес оновлення графічних елементів шляхом попереднього розрахунку змін у віртуальній пам'яті та їх селективного перенесення в реальну модель об’єктів документа. Такий підхід мінімізує обчислювальне навантаження на браузер кінцевого користувача, забезпечуючи високу продуктивність застосунку навіть при інтенсивному опрацюванні даних.

Для забезпечення передбачуваності станів системи в AiDocs реалізовано принцип **одностороннього потоку даних**. Це дозволяє здійснювати прецизійний контроль за змінами стану компонентів, що суттєво полегшує процеси налагодження та верифікації коду.

Позиціонування React у середовищі веб-розробки

Згідно з офіційною дефініцією, React є спеціалізованою бібліотекою для побудови користувацьких інтерфейсів, а не повнофункціональним фреймворком у класичному розумінні. Її універсальність підтверджується можливістю взаємодії з різними середовищами рендерингу: зокрема, у тандемі з **ReactDOM** бібліотека застосовується для створення веб-додатків, тоді як **React Native** дозволяє розробляти нативні мобільні рішення. Вживання терміна «фреймворк» щодо React зазвичай є розмовним узагальненням, що вказує на роль бібліотеки у вирішенні комплексних завдань фронтенд-розробки [5].

Основним концептуальним завданням React є декомпозиція інтерфейсу на автономні, логічно завершені фрагменти - компоненти. Модульна структура дозволяє розробникам концентруватися на проектуванні дизайну та логіки взаємодії, абстрагуючи низькорівневі процеси маніпуляції DOM-

деревом. На відміну від альтернативних фреймворків (наприклад, Angular), React не нав'язує жорстких конвенцій щодо організації файлової структури, надаючи розробникам гнучкість у виборі методології кодування.

Еволюційний контекст та практичне застосування

Поява React стала відповіддю на зростаючі потреби у високій продуктивності інтерактивних платформ, прикладом чого є соціальна мережа Facebook. В умовах опрацювання мільярдів одиниць контенту виникає необхідність у миттєвій реакції інтерфейсу на дії користувача (наприклад, реєстрація вподобань або додавання коментарів) без повного перезавантаження сторінки.

Історично розвиток веб-технологій пройшов шлях від статичного HTML та початкових інструментів маніпуляції DOM (JavaScript 1996 року, бібліотека jQuery) до потужних фреймворків. React радикально змінив підхід до створення контенту, фактично генеруючи HTML безпосередньо за допомогою JavaScript.

Технологічно цей процес реалізується через:

- **JSX (JavaScript XML):** синтаксичне розширення, що дозволяє описувати структуру інтерфейсу в межах логіки коду.
- **Babel:** транспайлер, що перетворює JSX-код у стандартний JavaScript, придатний для інтерпретації браузером.
- **Диференціацію станів:** при виникненні змін у конкретному компоненті система фіксує розбіжність між Virtual DOM та реальним HTML DOM, після чого здійснює «патчинг» лише тих елементів, що зазнали трансформації.

Завдяки поєднанню компонентно-орієнтованої архітектури та інтелектуального рендерингу, React залишається домінуючою технологією для розробки сучасних односторінкових застосунків (SPA), забезпечуючи високу якість користувацького досвіду в динамічному цифровому середовищі.

Порівняльний аналіз продуктивності Virtual DOM та традиційних методів маніпуляції об’єктами сторінки:

У класичній моделі розробки веб-інтерфейсів будь-яка зміна даних ініціює пряме звернення до **Real DOM**. Основна проблема цього підходу полягає в тому, що кожна модифікація одного вузла (наприклад, зміна текстового поля або оновлення лічильника) часто спричиняє каскадний перерахунок усієї геометрії сторінки (процеси *reflow* та *repaint*). Оскільки операції з DOM є найбільш ресурсозатратними в контексті браузерних обчислень, пряма маніпуляція об’єктами при роботі з динамічними даними призводить до значного зниження FPS (кількості кадрів на секунду) та візуальних затримок [6].

Впровадження **Virtual DOM** у межах бібліотеки React пропонує принципово інший алгоритм, який базується на мінімізації прямої взаємодії з браузерним API. Процес оптимізації реалізується через три основні етапи:

1. **Рендеринг у віртуальну пам'ять:** При зміні стану компонента React створює повну копію об'єктного дерева у віртуальному представленні. Ця операція виконується надзвичайно швидко, оскільки не задіює графічні ресурси браузера.
2. **Диференціація (Diffing):** Система порівнює поточну версію Virtual DOM із попередньою за допомогою спеціалізованого алгоритму узгодження (*reconciliation*). Це дозволяє точно ідентифікувати лише ті вузли, зміст яких фактично змінився.
3. **Пакетне оновлення (Batching):** Замість серії послідовних оновлень Real DOM, React формує єдиний пакет змін. Це дозволяє браузеру виконати перерахунок стилів та відображення лише один раз для цілої групи модифікацій.

2.2. Остання версія React.js та її особливості.

Вихід React 18 став визначальним етапом у розвитку бібліотеки, заклавши підґрунтя для кардинального покращення продуктивності веб-додатків та вдосконалення архітектурних підходів до розробки. Одним із найбільш фундаментальних нововведень є впровадження механізму паралельного рендерингу (Concurrent Mode). На відміну від попередніх ітерацій, де оновлення інтерфейсу відбувалися синхронно, що нерідко спричиняло блокування головного потоку при роботі зі складними структурами, паралельний режим базується на асинхронній моделі опрацювання завдань. Це дозволяє системі одночасно керувати декількома процесами рендерингу, інтелектуально визначаючи пріоритетність операцій та призупиняючи менш важливі обчислення на користь миттєвої реакції на дії користувача. Такий підхід забезпечує безперервну чуйність інтерфейсу в умовах високої інтерактивності та динамічного контенту [3].

Суттєвий прогрес було досягнуто і в автоматизації внутрішніх процесів бібліотеки, зокрема через механізм автоматичного пакетування (Automatic Batching). У попередніх версіях розробники часто стикалися з необхідністю ручної оптимізації оновлень станів для запобігання зайвим циклам перемальовування компонентів. React 18 інтелектуально групує декілька змін стану в єдиний цикл рендерингу, незалежно від того, де вони виникають — у межах обробників подій, промісів чи асинхронних запитів. Це не лише підвищує загальну продуктивність додатку, але й дозволяє створювати чистіший програмний код, звільняючи розробника від мікро-оптимізації оновлень на низькому рівні.

У контексті управління асинхронними потоками даних важливе місце посідає розширення функціоналу Suspense. В оновленій версії цей механізм дозволяє декларативно керувати станами очікування під час завантаження інформації. Використання Suspense дає змогу чітко розмежувати частини дерева компонентів, що потребують підготовки даних, та автоматизувати відображення резервного контенту або індикаторів завантаження. Це суттєво

спрощує координацію складних операцій та підвищує якість користувацького досвіду, усуваючи необхідність у розгалуженій логіці обробки станів «loading» та «error» у кожному окремому компоненті.

Паралельно з цим, у React 18 було вдосконалено архітектуру меж помилок (Error Boundaries). Спеціалізовані компоненти тепер забезпечують більш гнучкий контроль над ізоляцією критичних збоїв у дочірніх структурах, запобігаючи дестабілізації всього додатку. Це гарантує відмовостійкість системи та дозволяє реалізувати витончені сценарії відновлення працездатності інтерфейсу після виникнення непередбачуваних помилок.

Нарешті, оновлені клієнтські та серверні API рендерингу забезпечують безперебійну міграцію зі старих версій, дозволяючи розробникам поступово інтегрувати нові можливості при збереженні сумісності з режимом React 17.

Така архітектурна гнучкість сприяє створенню високопродуктивних рішень, що однаково ефективно функціонують як на стороні клієнта, так і в серверному середовищі [7].

Порівняльна характеристика React 17 та React 18

Перехід від сімнадцятої до вісімнадцятої версії бібліотеки React ознаменував фундаментальну зміну парадигми рендерингу: трансформацію з лінійної синхронної моделі у багатопотокову паралельну структуру. У межах архітектури React 17 процес оновлення інтерфейсу мав монолітний характер, що означало неможливість переривання циклу рендерингу до його повного завершення. Такий підхід створював суттєві перепони для продуктивності складних односторінкових застосунків, оскільки тривалі обчислення в основному потоці призводили до тимчасової втрати чуйності інтерфейсу та неможливості браузера оперативно обробляти події введення даних користувачем.

Впровадження React 18 базується на концепції Concurrent React, яка докорінно змінює механіку взаємодії з DOM. На відміну від попередньої версії, де розробники оперували послідовною чергою оновлень, нова архітектура дозволяє системі розбивати завдання на низькорівневі фрагменти,

що виконуються асинхронно. Це впроваджує інтелектуальну систему пріоритезації: критично важливі дії, як-от натискання клавіш або кліки, отримують вищий пріоритет і здатні тимчасово призупиняти фонові процеси рендерингу масивних компонентів. Після завершення обробки термінового запиту система автоматично повертається до перерваного завдання, що забезпечує стабільну частоту оновлення екрана та виключає візуальні затримки [8].

Суттєві трансформації торкнулися і механізмів ініціалізації програмного середовища. Якщо в React 17 використовувався традиційний метод `ReactDOM.render` для синхронного створення кореневого вузла, то у вісімнадцятій версії впроваджено API `createRoot`. Даний інструмент активує всі нові можливості бібліотеки за замовчуванням, дозволяючи системі ефективніше розподіляти обчислювальні ресурси процесора та оптимізувати використання оперативної пам'яті. Важливою відмінністю є також обсяг автоматичного пакетування (Batching): якщо раніше цей процес обмежувався лише обробниками подій React, то в новій версії він поширюється на всі асинхронні операції, включаючи проміси та таймери, що радикально зменшує кількість надлишкових циклів перемальовування[3].

Окремої уваги заслуговує еволюція серверного рендерингу (SSR). У React 17 цей процес мав статичний характер, вимагаючи повної підготовки контенту на сервері перед його передачею клієнту. React 18 пропонує стрімінгову модель передачі даних через механізм *Suspense*, що дозволяє надсилати готові частини сторінки поступово. Це значно скорочує час до першої візуалізації та підвищує загальну продуктивність системи AiDocs, забезпечуючи оптимальний баланс між функціональною насиченістю та швидкістю відгуку. Таким чином, перехід до нової архітектури є стратегічно важливим кроком для розробки сучасних інтелектуальних систем, орієнтованих на роботу з великими обсягами динамічної інформації [9].

2.3. Node.js кросплатформенне середовище.

Node.js постає як детерміноване кросплатформове середовище виконання JavaScript із відкритим вихідним кодом, що функціонує на базі високопродуктивного двигуна Google V8. Поява цієї технології радикально змінила парадигму використання мови JavaScript, вивівши її за межі клієнтської сторони браузера та заклавши фундамент для розробки високонавантажених серверних рішень. Ефективність Node.js базується на унікальній архітектурі, яка нівелює архаїчні обмеження традиційних синхронних середовищ.

Фундаментальну основу архітектури Node.js складають два взаємопов'язані принципи: неблокуюча модель операцій вводу/виводу (**Non-blocking I/O**) та події-орієнтована асинхронна парадигма. Механізм неблокуючого вводу/виводу дозволяє середовищу здійснювати безперебійну обробку клієнтських запитів, не очікуючи завершення тривалих операцій (наприклад, звернень до бази даних або файлової системи), що забезпечує паралельне виконання завдань без перешкод для основного потоку виконання. Це суттєво підвищує коефіцієнт корисної дії системи та гарантує високу швидкість реагування додатку [10].

Асинхронна парадигма логічно інтегрується в подієво-керовану сутність JavaScript. Завдяки ефективному використанню функцій зворотного виклику (callbacks) та циклу подій (Event Loop), Node.js досягає оптимального розподілу апаратних ресурсів. Це дозволяє створювати масштабовані веб-додатки, здатні витримувати значні навантаження при відносно низькому споживанні пам'яті [11].

Ключові переваги використання Node.js у сучасній веб-розробці

Широке впровадження Node.js у професійну розробку зумовлене комплексом стратегічних переваг, що охоплюють як технічні, так і організаційні аспекти:

• **Продуктивність та масштабованість.** Завдяки компіляції JavaScript у машинний код за допомогою двигуна V8 та використанню моделі, керованої

подіями, Node.js демонструє високу швидкість виконання операцій. Архітектура середовища дозволяє легко масштабувати додатки горизонтально та вертикально, що є критично важливим для сервісів із великою кількістю одночасних з'єднань.

• **Уніфікація технологічного стеку.** Можливість використання JavaScript як на клієнтській, так і на серверній стороні (Full-stack JS) забезпечує цілісність розробки. Це спрощує комунікацію всередині команди, прискорює цикли розробки та дозволяє повторно використовувати бізнес-логіку між фронтендом і бекендом.

• **Розвинена екосистема модулів.** Через систему керування пакетами **npm**, Node.js надає доступ до найбільшого у світі репозиторію бібліотек із відкритим вихідним кодом. Це стимулює модульний підхід до розробки, дозволяючи інтегрувати готові рішення та уникати надлишкового написання коду для типових завдань.

• **Оптимізація для мікросервісної архітектури.** Легковажність та модульний дизайн роблять Node.js ідеальним вибором для побудови мікросервісів. Складні системи легко декомпонуються на автономні, слабко пов'язані сервіси, що значно підвищує відмовостійкість та гнучкість управління програмним продуктом.

Підсумовуючи, Node.js слід розглядати не лише як середовище виконання, а як цілісну екосистему, адаптовану до динамічних вимог сучасної інженерії програмного забезпечення. Її використання в розробці інтелектуальних систем управління документацією, таких як **AiDocs**, дозволяє забезпечити необхідну швидкість обробки даних та гнучкість архітектури [12].

Висновки до розділу.

Ефективність функціонування сучасних систем управління документацією безпосередньо залежить від обраного інженерного підходу, який має поєднувати високу швидкість обробки даних із гнучкістю інтерфейсу. Використання бібліотеки React як фронтенд-інструменту дозволяє реалізувати компонентно-орієнтовану архітектуру, що спрощує масштабування системи та забезпечує високу реактивність завдяки механізму Virtual DOM. Впровадження інновацій версії React 18, зокрема паралельного рендерингу та автоматичного пакетування оновлень, дозволяє мінімізувати навантаження на клієнтську сторону та гарантувати плавність взаємодії навіть при опрацюванні великих масивів динамічного контенту.

Вибір середовища Node.js для реалізації серверної логіки є обґрунтованим з огляду на його асинхронну природу та неблокуючу модель вводу/виводу. Це забезпечує високу пропускну здатність системи при обробці великої кількості одночасних запитів, що є критично важливим для сервісів електронного документообігу в умовах інтенсивної експлуатації. Крім того, уніфікація мови програмування (JavaScript) на всіх рівнях розробки сприяє цілісності архітектури, спрощує підтримку коду та прискорює впровадження нових функціональних можливостей.

Інтеграція розглянутих технологій створює надійний фундамент для побудови системи, що відповідає сучасним вимогам до продуктивності, безпеки та масштабованості. Використання модульного підходу та сучасних методів рендерингу дозволяє не лише автоматизувати рутинні операції зі створення документів, а й закласти потенціал для подальшого впровадження інтелектуальних алгоритмів аналізу даних. Таким чином, обраний технологічний комплекс є оптимальним для реалізації мети дипломної роботи та створення конкурентоспроможного програмного продукту.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Розроблення системи створення шаблонів.

Програмна система AiDocs постає як інтелектуальне середовище, спроектоване для комплексної автоматизації процесів генерації та адміністрування цифрової документації. Технологічний базис платформи, реалізований на основі фреймворку React, зумовлює високу швидкість відгуку інтерфейсу та стабільність функціонування динамічних компонентів. Використання компонентно-орієнтованого підходу дозволяє структурувати складний функціонал системи у вигляді автономних модулів, що забезпечує ергономічність взаємодії та спрощує процес освоєння інструментарію кінцевим користувачем.

Центральним елементом прикладного значення AiDocs є механізм інтелектуального проектування шаблонів. Система надає можливість оперативного створення нових структурованих форм на основі верифікованих прототипів, що значно підвищує ефективність підготовки специфічної документації. Розгалужена бібліотека пресетів дозволяє здійснювати точний підбір і кастомізацію документів відповідно до індивідуальних запитів суб’єкта. Завершальний етап опрацювання даних передбачає можливість виведення сформованих об’єктів на друк або їх конвертацію у кросплатформний формат PDF, що гарантує збереження автентичності та цілісності інформації [13].

Важливим аспектом архітектури AiDocs є дотримання протоколів інформаційної безпеки та розмежування прав доступу. Повний функціонал системи, пов'язаний із безпосереднім опрацюванням та завантаженням файлів, доступний виключно авторизованим користувачам, що мінімізує ризики несанкціонованого використання конфіденційних даних. Водночас для незареєстрованих суб’єктів передбачено демонстраційний режим, який дозволяє візуалізувати наявні шаблони у формі захищених водяними знаками

зображень. Це дає змогу провести попередню оцінку функціонального різноманіття та потенціалу платформи без порушення цілісності інтелектуальної власності.

У підсумку, AiDocs слід розглядати як багатофункціональну екосистему, яка виходить за межі звичайного текстового редактора. Вона інтегрує в собі сучасні інструменти для продуктивного управління документаційним фондом, поєднуючи високу технологічність виконання з практичною орієнтованістю на оптимізацію управлінських та діловодних процесів (рис. 3.1).

Рис. 3.1. Бібліотека React.

Архітектурна побудова системи AiDocs базується на принципах компонентного підходу, що передбачає декомпозицію веб-інтерфейсу на автономні модулі. Таке рішення забезпечує високий рівень реюзабільності (повторного використання) та модифікації окремих елементів, що суттєво спрощує процеси розробки, технічного обслуговування та колективної роботи над проєктом [3].

Однією з фундаментальних переваг застосування бібліотеки React у межах AiDocs є механізм віртуального DOM (Virtual DOM). Ця технологія забезпечує високопродуктивне відображення змін в інтерфейсі, мінімізуючи навантаження на обчислювальні ресурси браузера користувача та гарантуючи стабільну роботу додатка навіть при опрацюванні великих масивів даних.

Для забезпечення консистентності та передбачуваності стану системи використано принцип односпрямованого потоку даних. Такий підхід дозволяє ефективно відстежувати зміни у станах компонентів, що спрощує процес налагодження (debugging) та підвищує загальну відмовостійкість програмного продукту.

Завдяки використанню хуків, зокрема useState та useEffect, реалізовано динамічне реагування на події та оновлення даних у реальному часі, що є критично важливим для функціонування інтерактивних елементів платформи.

Розширюваність системи досягається шляхом інтеграції додаткових бібліотек, таких як Redux для глобального керування станом та React Router для побудови розгалуженої навігації. Використання широкої екосистеми готових компонентів та стилізаційних інструментів дозволило оптимізувати процес розробки та забезпечити сучасний візуальний вигляд інтерфейсу.

Отже, інтелектуальна система AiDocs позиціонується як інноваційне рішення для автоматизації управління бізнес-документацією.

Поєднання компонентної архітектури, декларативного підходу до опису інтерфейсів та потужних інструментів управління станом дозволило створити швидку, надійну та ергономічну платформу, що відповідає актуальним потребам сучасного підприємництва [14].

Процес імплементації системи включав кілька послідовних етапів, спрямованих на забезпечення технологічної цілісності та успішного розгортання Web-проєкту. (рис. 3.2).

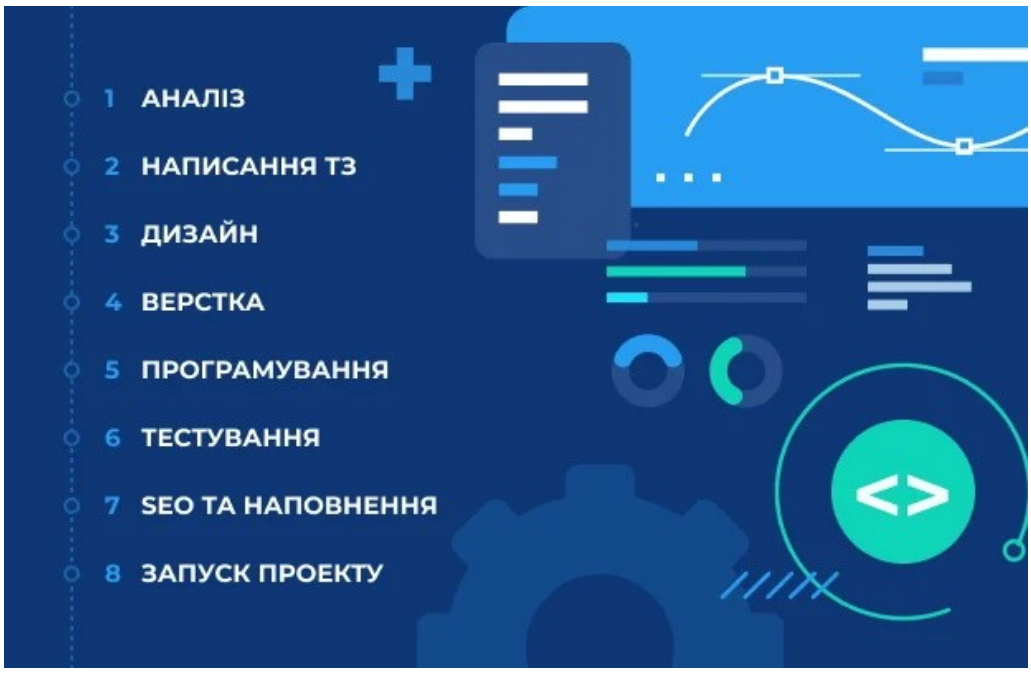


Рис.2 Етапи розробки

Етапи проєктування та реалізації інтелектуальної системи

Процес створення інтелектуальної системи керування документацією базується на системному підході та охоплює кілька послідовних стадій, кожна з яких є критично важливою для забезпечення якості програмного продукту.

1. **Передпроєктний аналіз та визначення вимог.** На початковому етапі проводиться детальне дослідження предметної області та чітке формулювання мети розробки. Визначається цільова аудиторія, ключові бізнес-процеси, які потребують автоматизації, та необхідний функціональний набір. Результатом аналізу є формування концепції системи та визначення очікуваних показників її ефективності.

2. **Розробка технічного завдання (ТЗ).** На основі отриманих даних здійснюється планування логічної архітектури та **проєктування** структури вебсайту. Формується технічне завдання, що містить детальний опис вимог до апаратного забезпечення, безпеки, інтерфейсу та алгоритмів опрацювання бізнес-документів.

3. **Проєктування інтерфейсу та UI/UX дизайн.** Етап включає створення прототипів, візуальних макетів та скетчів сторінок. Використання дизайн-системи дозволяє вибудувати єдину стилістичну концепцію та

ергономічне розташування елементів керування, що забезпечує зручність взаємодії користувача з платформою.

4. **Фронтенд-розробка та верстка.** На цьому етапі реалізується візуальна частина системи. Використання мов розмітки **HTML5** та каскадних таблиць стилів **CSS3** (або відповідних фреймворків) дозволяє трансформувати графічні макети у функціональний інтерфейс, забезпечуючи адаптивність та коректне відображення на різних пристроях.

5. **Програмна реалізація та логіка системи.** За допомогою бібліотеки **React** вебсайт наповнюється інтерактивним функціоналом. На цьому етапі розробляється програмна логіка, створюються компоненти для обробки документів, впроваджуються механізми управління станом додатка та налаштовується взаємодія з серверною частиною.

6. **Верифікація та тестування.** Після завершення активної фази розробки проводиться комплексне тестування системи. Воно охоплює перевірку працездатності всіх модулів, пошук багів та валідацію відповідності реалізованого функціоналу вимогам ТЗ. Тестування здійснюється як ручним методом, так і з використанням засобів автоматизації для перевірки критичних сценаріїв.

7. **Розгортання (Deployment).** Успішно протестована система розміщується на серверних потужностях або хмарних хостинг-платформах. Це забезпечує публічний доступ до ресурсу через мережу Інтернет та стабільне функціонування під навантаженням.

8. **SEO-оптимізація та контентне наповнення.** Проводяться заходи з пошукової оптимізації для покращення індексації сайту. Паралельно здійснюється початкове наповнення бази даних шаблонами бізнес-документів, нормативною інформацією та іншим необхідним контентом для старту роботи користувачів.

9. **Експлуатація та супровід системи.** Фінальний етап передбачає постійний моніторинг роботи платформи, виправлення виявлених у процесі

експлуатації помилок, оновлення програмних залежностей та вдосконалення функціоналу відповідно до нових вимог ринку.

Структура проєкту та використаний інструментарій

Під час реалізації інтелектуальної системи на базі бібліотеки React було сформовано чітку ієрархію файлової структури (рис. 3.3), що забезпечує логічне розділення рівнів представлення, логіки та управління станом.

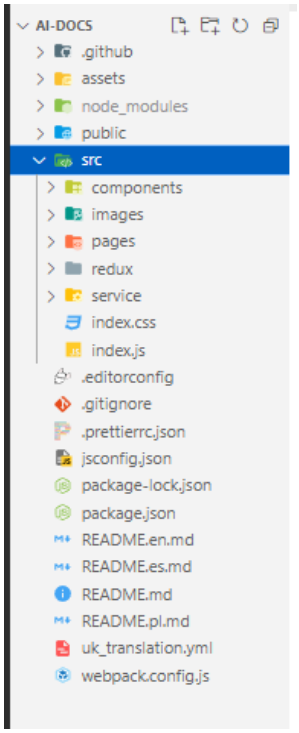


Рис. 3.3. Ієрархія файлової структури

Основні директорії та файли проєкту включають:

- src/index.js. Вхідна точка додатка, яка відповідає за ініціалізацію React-програми та рендеринг кореневого компонента в DOM-дерево вебсторінки. У цьому файлі здійснюється зв’язок між логічною структурою додатка та його фізичним відображенням у браузері.
- src/App.js. Головний компонент-контейнер, що виступає основою для побудови ієрархії інших модулів системи. Він слугує точкою збірки для всіх функціональних частин інтерфейсу платформи.
- src/components. Директорія, що містить набір компонентів користувацького інтерфейсу. Кожен модуль ізольований у межах окремого файлу, що спрощує тестування, модифікацію та повторне використання коду.

- src/services. Спеціалізований каталог для розміщення логіки взаємодії із зовнішніми API та серверною частиною (зокрема Firebase). Відокремлення мережевих запитів у цей рівень дозволяє розмежувати бізнес-логіку та візуальне представлення, підвищуючи чистоту коду.
- src/store. Директорія, що містить конфігурацію стейт-менеджера (наприклад, Redux). Тут зосереджені редьюсери (reducers), дії (actions) та селектори, що забезпечують централізоване управління станом і гарантують передбачувану поведінку додатка при складних транзакціях даних.

Для забезпечення ефективного функціонування розробленої системи було задіяно наступні інструменти:

ReactDOM — фундаментальний пакет, що забезпечує механізм рендерингу React-компонентів у реальний DOM. Даний інструмент виступає сполучною ланкою між програмними об’єктами бібліотеки та об’єктною моделлю документа в браузері. Завдяки використанню алгоритмів порівняння (diffing), ReactDOM оптимізує процес оновлення сторінки, забезпечуючи високу швидкість відображення змін без необхідності повного перевантаження інтерфейсу.

React Router — спеціалізована бібліотека, призначена для реалізації механізмів маршрутизації в екосистемі React. Вона надає інструментарій для декларативного опису шляхів (маршрутів) та їхнього зіставлення з відповідними функціональними компонентами, що підлягають візуалізації. Використання **React Router** дозволяє проєктувати багасторінкові інтерфейси в межах концепції односторінкових додатків (SPA), забезпечуючи гнучке налаштування навігації. Бібліотека підтримує роботу з параметрами динамічних маршрутів, вкладеними структурами та інтеграцію з історією переглядів браузера, що забезпечує коректну роботу кнопок навігації користувача (рис. 4).

Redux є потужним інструментом для централізованого керування станом складних програмних систем. В основі бібліотеки лежить архітектура **Flux**, яка пропонує сувору логіку контролю над глобальними даними додатка.

Фундаментальним принципом **Redux** є використання єдиного об’єкта — сховища (**store**), у якому акумулюється повний стан системи.

```
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

Взаємодія компонентів із даними відбувається через звернення до сховища для отримання актуального стану або шляхом ініціації дій (**actions**), що сигналізують про необхідність внесення змін. Такий підхід гарантує передбачуваність поведінки інтерфейсу, полегшує процеси відлагодження та забезпечує прозорість життєвого циклу даних у межах всього проєкту. (рис 3.5).

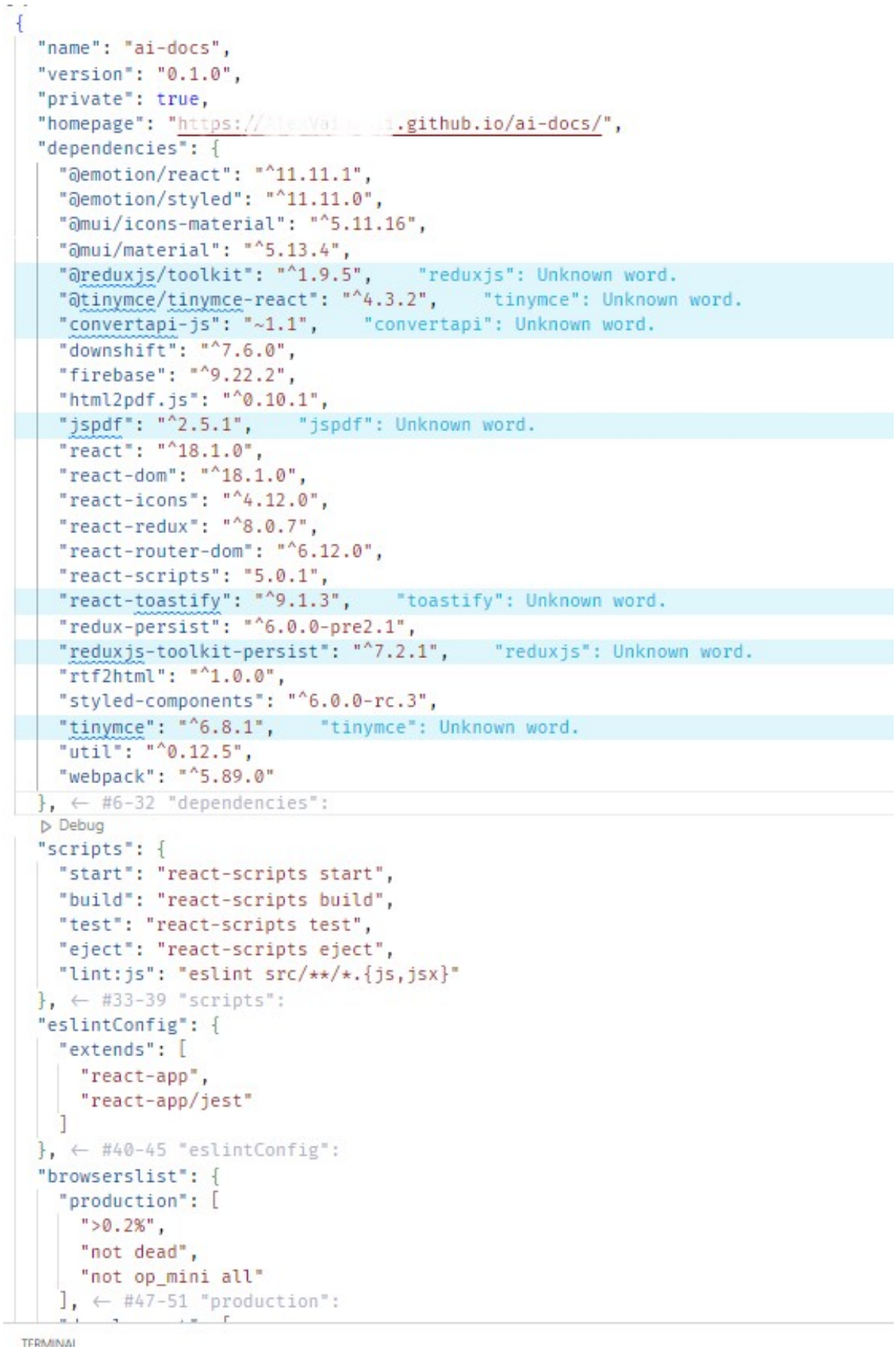
Файл **package.json** є фундаментальним елементом архітектури будь-якого **проєкту**, розробленого в середовищі **Node.js**, зокрема й для додатків на базі **React**. Даний файл виконує роль маніфесту, що містить вичерпні метадані про **проєкт**, визначає перелік необхідних зовнішніх залежностей (бібліотек) та зберігає ключові конфігураційні налаштування [15].

```
} 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. 3.5. Redux

Завдяки **package.json** забезпечується автоматизація управління життєвим циклом програмного продукту, включаючи визначення версій пакетів, налаштування скриптів для збірки, тестування та запуску застосунку.

Це гарантує відтворюваність **проєктного** середовища на різних етапах розробки та розгортання, що є критично важливим для стабільної роботи інтелектуальної системи (рис 6).



```
{
  "name": "ai-docs",
  "version": "0.1.0",
  "private": true,
  "homepage": "https://[illegible].github.io/ai-docs/",
  "dependencies": {
    "@emotion/react": "^11.11.1",
    "@emotion/styled": "^11.11.0",
    "@mui/icons-material": "^5.11.16",
    "@mui/material": "^5.13.4",
    "@reduxjs/toolkit": "^1.9.5",    "reduxjs": Unknown word.
    "@tinymce/tinymce-react": "^4.3.2",    "tinymce": Unknown word.
    "convertapi-js": "~1.1",    "convertapi": Unknown word.
    "downshift": "^7.6.0",
    "firebase": "^9.22.2",
    "html2pdf.js": "^0.10.1",
    "jspdf": "^2.5.1",    "jspdf": Unknown word.
    "react": "^18.1.0",
    "react-dom": "^18.1.0",
    "react-icons": "^4.12.0",
    "react-redux": "^8.0.7",
    "react-router-dom": "^6.12.0",
    "react-scripts": "5.0.1",
    "react-toastify": "^9.1.3",    "toastify": Unknown word.
    "redux-persist": "^6.0.0-pre2.1",
    "reduxjs-toolkit-persist": "^7.2.1",    "reduxjs": Unknown word.
    "rtf2html": "^1.0.0",
    "styled-components": "^6.0.0-rc.3",
    "tinymce": "^6.8.1",    "tinymce": Unknown word.
    "util": "^0.12.5",
    "webpack": "^5.89.0"
  }, ← #6-32 "dependencies":
  ▷ Debug
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject",
    "lint:js": "eslint src/**/*.{js,jsx}"
  }, ← #33-39 "scripts":
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  }, ← #40-45 "eslintConfig":
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ], ← #47-51 "production":
```

TERMINAL

Рис. 3.6. Package.json

Styled Components — це високоефективна бібліотека, що реалізує концепцію **CSS-in-JS** для стилізації компонентів у середовищі **React**. На

відміну від традиційного використання зовнішніх каскадних таблиць стилів, даний інструмент дозволяє інтегрувати описи візуального представлення безпосередньо в програмний код компонента.

Завдяки автоматичній генерації унікальних CSS-класів для кожного елемента, бібліотека забезпечує високий рівень ізоляції стилів, що унеможливлює виникнення конфліктів іменування в межах великих проєктів.

Фундаментальною перевагою **Styled Components** є нативна підтримка динамічної стилізації. Це дозволяє адаптувати зовнішній вигляд інтерфейсу залежно від вхідних параметрів (props) або внутрішнього стану додатка, що є критично важливим для створення гнучких та реактивних користувацьких інтерфейсів.

Окрім цього, інструментарій бібліотеки спрощує впровадження систем глобального тематичного оформлення (theming). Це дозволяє централізовано керувати колірними схемами, типографікою та іншими стилістичними константами, забезпечуючи візуальну цілісність платформи.

Також **Styled Components** надає зручні механізми для опису декларативних анімацій, що дозволяє створювати динамічні та інтерактивні елементи, підвищуючи загальну ергономіку та привабливість програмного продукту (рис 7).

Firebase — це багатофункціональна хмарна платформа від Google, призначена для розробки високоефективних Web- та мобільних застосунків.

Вона надає комплексну інфраструктуру та широкий спектр сервісів, що дозволяють оптимізувати процес створення складних програмних систем.

```
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

До ключових можливостей **Firebase**, використаних у межах даного **проєкту**, належать:

- **Синхронізація даних у реальному часі.** Завдяки використанню **Firebase Realtime Database**, система забезпечує автоматичну синхронізацію інформації між усіма підключеними клієнтами. Це дозволяє реалізувати миттєву реакцію інтерфейсу на зміни в базі даних без необхідності додаткових запитів.

- **Автентифікація користувачів.** Вбудований сервіс **Firebase Authentication** спрощує процеси ідентифікації та авторизації. Платформа підтримує різноманітні методи доступу, зокрема через електронну пошту та соціальні мережі, гарантуючи при цьому високий рівень безпеки персональних даних.
- **Firebase Cloud Messaging (FCM).** Даний інструментарій дозволяє реалізувати розсилку повідомлень на мобільні пристрої та вебзастосунки в режимі реального часу, що є критично важливим для оперативного сповіщення користувачів та підвищення рівня їхньої взаємодії з платформою.
- **Хостинг та мережа доставки контенту (CDN).** Інтегровані засоби хостингу в поєднанні з глобальною мережею **CDN** забезпечують швидке розгортання застосунку та мінімальну затримку при завантаженні ресурсів незалежно від географічного розташування користувача.
- **Хмарні функції (Cloud Functions).** Платформа надає можливість розгортання серверного коду в безсерверному середовищі. Це дозволяє виконувати складні обчислення на стороні сервера та інтегрувати сторонні сервіси, зберігаючи архітектуру фронтенду лаконічною.
- **Аналітика та моніторинг.** Інструментарій для відстеження активності користувачів надає можливість аналізувати поведінкові сценарії, що сприяє подальшій оптимізації функціональності та покращенню користувацького досвіду (UX).

Інтеграція **Firebase** із **React**-додатком здійснюється за допомогою спеціалізованого **SDK для JavaScript**. Це забезпечує зручний програмний інтерфейс для взаємодії з хмарними сервісами безпосередньо з коду компонентів. Використання даної платформи дозволяє делегувати складні завдання з управління інфраструктурою, зосередивши основну увагу на реалізації унікальної логіки та функціональних можливостей інтелектуальної системи. (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. 3.8. Firebase ініціалізування в проекті

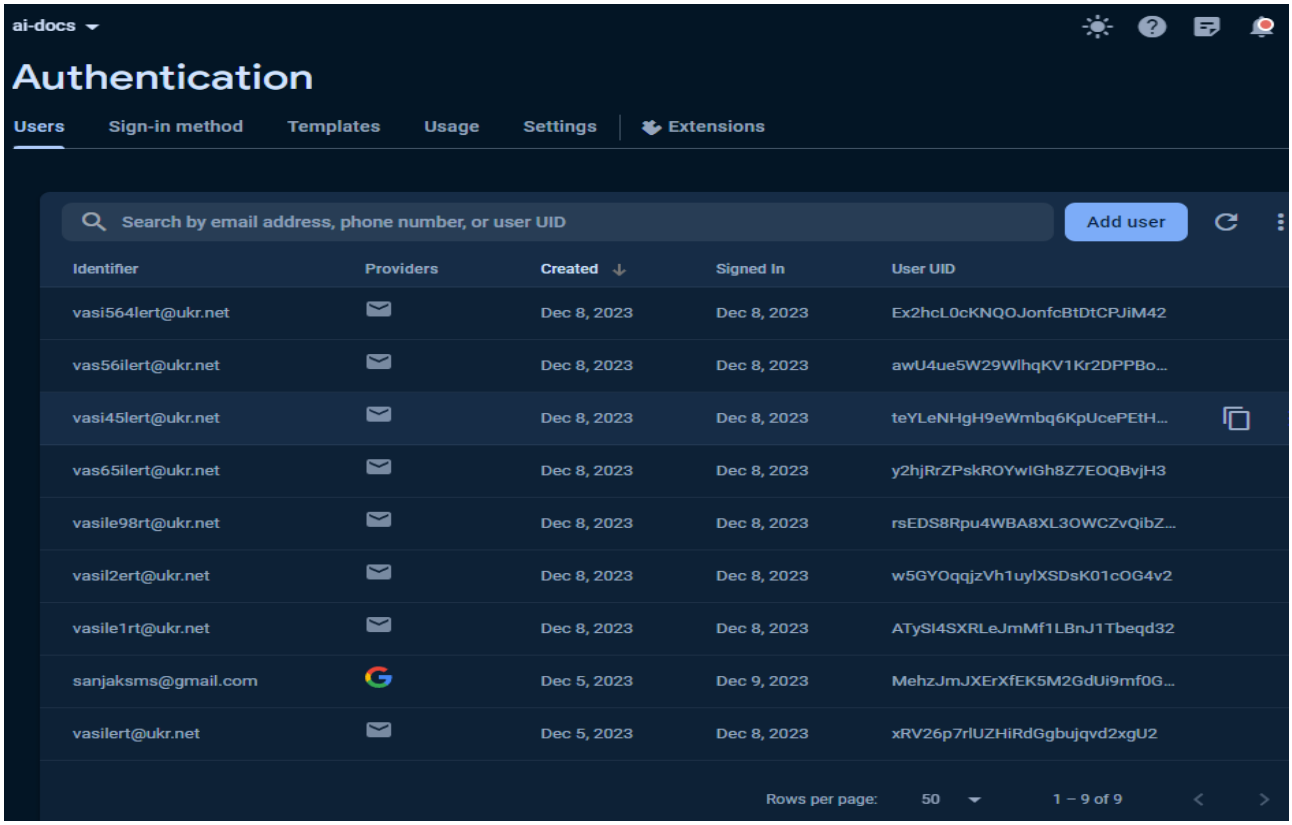


Рис. 3.9. Firebase Аутентифікація користувачів

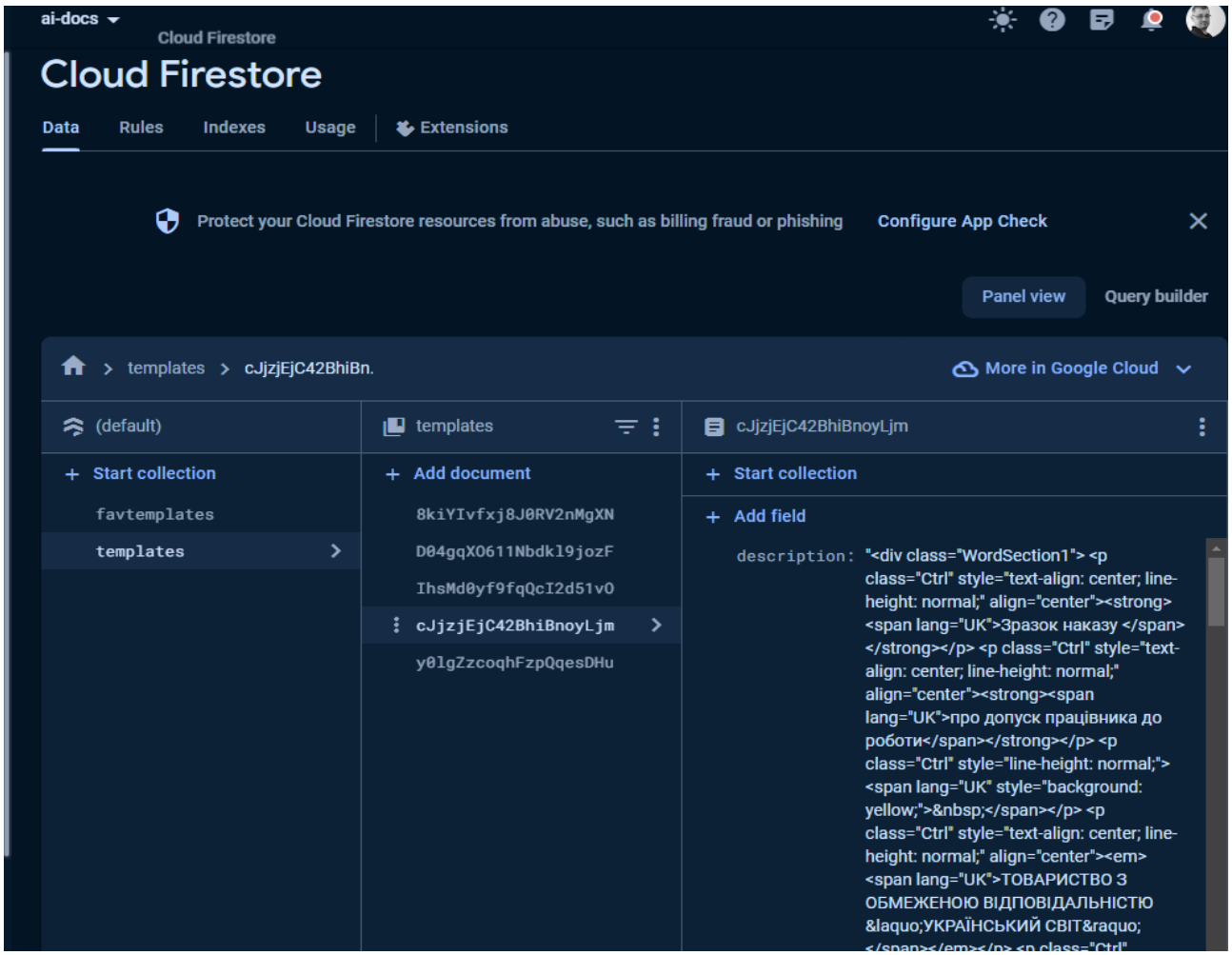


Рис. 3.10. Firebase Збереження даних на сервері

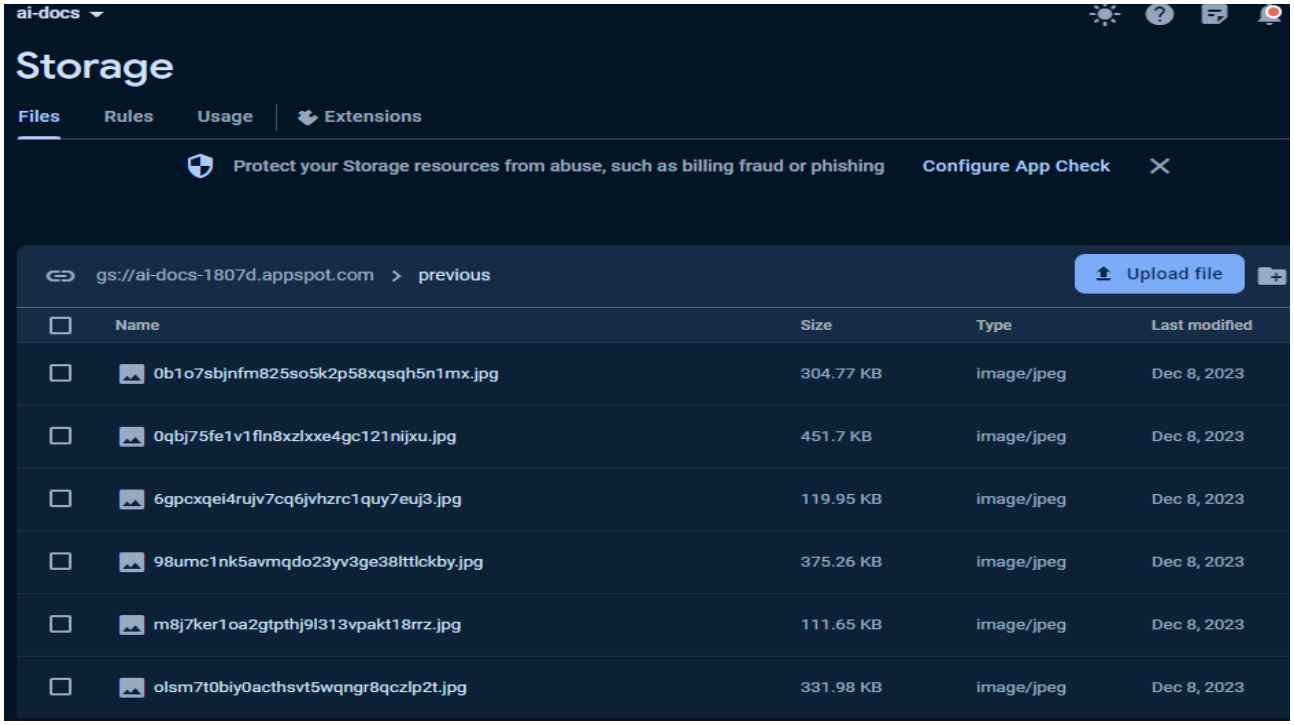


Рис. 3.11. Firebase Збереження файлів на сервері

```
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. 3.12. 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. 3.13. 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. 3.14. Firebase Операції додавання даних до бази

Реалізація модуля автентифікації користувачів

Використання зазначеного стеку бібліотек та інструментів суттєво оптимізує процес проєктування вебзастосунків на базі **React**, забезпечуючи розробнику розширений інструментарій для створення чистого, ефективного та масштабованого коду. Це не лише прискорює цикл розробки, а й гарантує високу підтримуваність програмного продукту в майбутньому.

Одним із пріоритетних завдань у межах роботи стала розробка авторського компонента автентифікації. Даний модуль реалізовано з використанням бібліотек **React** та **Redux**, що дозволило інтегрувати складну логіку управління станом користувача безпосередньо в архітектуру системи[16].

До ключових функціональних можливостей розробленого компонента належать:

• **Інтерфейс введення даних:** реалізація ергономічних форм для опрацювання облікових даних (електронної пошти та пароля);

• **Управління доступом:** забезпечення механізмів реєстрації нових користувачів та авторизації наявних у базі даних;

• **Інтеграція OAuth:** підтримка методів автентифікації через сторонні сервіси, зокрема через обліковий запис **Google**;

• **Зворотний зв'язок:** система динамічного відображення статусних повідомлень про успішне завершення операцій або виникнення помилок валідації;

• **Профіль користувача:** візуалізація персоналізованої інформації та реалізація безпечного виходу із системи;

• **Адаптивність режимів:** можливість гнучкого перемикання між формою автентифікації та іншими функціональними станами компонента.

Розроблений компонент характеризується високим рівнем гнучкості, що дозволяє інтегрувати його в будь-який розділ платформи, де виникає потреба в ідентифікації користувача.

Завдяки реалізації на основі сучасних методологій Web-проєктування та використанню найкращих практик безпеки, цей модуль виступає фундаментальним елементом системи, що забезпечує надійний захист конфіденційної інформації та коректне розмежування прав доступу (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. 3.15. 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. 3.16. Код відображення форми аутентифікації

Використання Redux Toolkit для управління станом системи

Для забезпечення ефективної автентифікації та централізованого збереження даних у проєкті було застосовано бібліотеку Redux Toolkit (RTK). Цей інструментарій є сучасним стандартом для керування станом додатків на базі React, оскільки він оптимізує архітектуру Redux, спрощуючи взаємодію з даними в межах усієї системи.

Redux Toolkit забезпечує структурований підхід до організації стану, пропонуючи єдине джерело істини для всіх компонентів застосунку. Завдяки інтеграції таких механізмів, як Redux Thunk, бібліотека дозволяє ефективно опрацьовувати асинхронні операції та побічні ефекти. Ключові переваги використання Redux Toolkit у розробленій системі включають:

• Оптимізація синтаксису. Використання функцій createSlice та configureStore дозволяє суттєво зменшити обсяг шаблонного коду (boilerplate). Це спрощує оголошення дій (actions) та редюсерів, роблячи структуру проєкту більш лаконічною та зрозумілою.

• Нативна підтримка асинхронності. Завдяки вбудованому проміжному програмному забезпеченню (middleware), зокрема Redux Thunk, реалізовано зручну взаємодію з Firebase API. Це дозволяє безперешкодно виконувати запити на автентифікацію та синхронізацію даних із сервером у фоновому режимі.

• Інструментарій для відлагодження. Повна сумісність із Redux DevTools надає розробнику можливість моніторингу змін стану в реальному часі. Це полегшує процес тестування, дозволяючи відстежувати кожну дію та її вплив на глобальний стан системи.

• Модульна архітектура. Redux Toolkit сприяє поділу логіки на окремі слайси (slices). Кожен модуль містить власний набір логіки та селекторів, що підвищує масштабованість додатка та спрощує паралельну розробку різних функціональних блоків.

Завдяки впровадженню Redux Toolkit, компонент автентифікації отримав можливість ефективно керувати сесіями користувачів, надійно

зберігати персональні дані та оперативно взаємодіяти з сервісами Firebase. Це забезпечує високу продуктивність управління станом, роблячи платформу стабільною та готовою до подальшого розширення функціоналу. (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. 3.17. 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. 3.18. «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. 3.19. 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. 3.20.«Слайс» на вибірку, додавання, видалення та обновлення даних у Firebase

Використання текстового редактора TinyMCE в архітектурі системи

Для забезпечення розширених можливостей редагування текстового контенту в межах проєкту було інтегровано WYSIWYG-редактор TinyMCE, адаптований для середовища React. Цей інструментарій надає користувачам інтуїтивно зрозумілий інтерфейс для професійної обробки тексту, що є критично важливим для створення та корегування бізнес-документації.

Ключові функціональні характеристики та переваги інтегрованого редактора включають:

• Розширений інструментарій форматування. TinyMCE пропонує багатофункціональну панель інструментів, що підтримує як базові операції (напівжирне накреслення, курсив, підкреслення), так і складні методи структурування контенту (марковані та нумеровані списки, багаторівневе вирівнювання тексту тощо).

• Опрацювання мультимедійного контенту. Редактор забезпечує нативну підтримку інтеграції зображень та відеоматеріалів. Користувачам доступні засоби редагування атрибутів медіафайлів, включаючи зміну розміру, кадрування та налаштування обтікання текстом.

• Робота зі структурованими даними. Реалізовано повноцінну підтримку таблиць для організації складних масивів інформації, а також можливість вставки та форматування специфічних текстових регіонів (контейнерів), що дозволяє чітко розмежовувати різні логічні блоки документа.

• Локалізація та багатомовність. Архітектура редактора підтримує гнучке налаштування мовних пакетів, що дозволяє адаптувати інтерфейс під потреби різних груп користувачів, забезпечуючи високий рівень інклюзивності.

• Модульність та використання плагінів. Завдяки плагінній системі функціональність редактора може бути розширена відповідно до специфічних вимог проєкту. Це дозволяє інтегрувати додаткові сервіси (наприклад, перевірку орфографії або вставку спецсимволів) без надмірного обтяження основного коду системи.

• Кросплатформеність та сумісність. Програмне рішення є повністю адаптивним, що гарантує коректну роботу редактора на пристроях із різною діагоналлю екрана та в усіх сучасних веббраузерах. Відкритий вихідний код бібліотеки дозволяє розробникам модифікувати функціонал та оптимізувати його під конкретні завдання автоматизації документообігу.

Впровадження TinyMCE значно підвищує ергономіку платформи, надаючи користувачам потужний інструмент для швидкого та якісного керування текстовим вмістом без необхідності володіння навичками верстки. (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. 3.21. Код редактору шаблону

Функціонал генерації та конвертації документів

Важливою особливістю інтегрованого редактора TinyMCE у межах даної системи є можливість прямого виведення створених шаблонів на друк. Окрім цього, архітектура платформи передбачає інструментарій для завантаження наявних шаблонів із подальшим їх редагуванням, що забезпечує гнучкість в управлінні документообігом.

Для забезпечення коректного перетворення даних між різними форматами та генерації звітів було задіяно бібліотеки convertapi-js та html2pdf.js. Синергія цих інструментів дозволяє автоматизувати процес

створення документів, роблячи формування вихідних матеріалів максимально продуктивним.

• html2pdf.js — це спеціалізований інструментарій для генерації PDF-файлів на основі вмісту вебсторінок. Бібліотека дозволяє трансформувати структурований HTML-код та відповідні CSS-стилі у формат PDF, зберігаючи візуальну цілісність документа. Вона надає розширені можливості керування параметрами вихідного файлу, зокрема:

- налаштування формату та орієнтації сторінок;
- контроль якості графічних об’єктів;
- встановлення полів та відступів.

• Інтеграція з React. Бібліотека html2pdf ефективно взаємодіє з компонентами системи, дозволяючи експортувати як окремі функціональні елементи, так і цілі сторінки додатка. Це забезпечує користувачам можливість миттєво отримувати готові до використання документи із заданими специфікаціями.

Застосування convertapi-js у поєднанні з html2pdf розширює можливості системи щодо обробки різноманітних файлових форматів, що перетворює AiDocs на потужну платформу для комплексної роботи з бізнес-документацією. (рис 3.22).

```
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. 3.22. Використання html2pdf

Convertapi-js — це спеціалізований інструментарій розробки (SDK) для мови програмування JavaScript, що забезпечує програмну взаємодію з хмарним сервісом ConvertAPI. Даний сервіс спеціалізується на високоточній конвертації файлів між різними форматами. У межах реалізованого проєкту

бібліотека використовується як ключовий інструмент для трансформації документів формату .docx у формати HTML (для подальшого редагування в інтерфейсі) та JPG (для візуального попереднього перегляду).

Використання convertapi-js у поєднанні з React дозволяє винести складні обчислювальні процеси конвертації на зовнішні серверні потужності, що суттєво знижує навантаження на клієнтську частину додатка. Бібліотека надає надійний інтерфейс для обробки файлових потоків, гарантуючи коректне збереження структури, стилів та форматування вихідних документів під час перетворення. Це забезпечує стабільну роботу системи при маніпуляціях із різними типами контенту та розширює функціональні можливості платформи щодо інтеграції з офісними форматами документів (рис 3.23).

```
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. 3.23. Використання convertapi-js

Механізми взаємодії з нереєстрованими користувачами

Для забезпечення відкритості системи та стимулювання залучення нових користувачів, у проєкті реалізовано функціонал попереднього перегляду шаблонів для неавторизованих відвідувачів. Дане рішення дозволяє ознайомитися з архітектурою та змістовим наповненням доступних бланків без необхідності негайного проходження процедури реєстрації.

Ключові аспекти даного підходу включають:

• **Візуалізація контенту.** Нереєстровані користувачі мають доступ до графічних копій шаблонів, що дозволяє оцінити структуру документа, його візуальне оформлення та відповідність конкретним бізнес-потребам ще на етапі ознайомлення.

• Захист авторських прав. З метою запобігання несанкціонованому використанню інтелектуальної власності, усі зображення для попереднього перегляду містять водяні знаки (watermarks). Це забезпечує надійний контроль над розповсюдженням контенту та захищає майнові права розробників шаблонів.

• Оптимізація користувацького досвіду (UX). Надання вільного доступу до каталогу сприяє формуванню лояльності аудиторії. Користувачі отримують можливість здійснити усвідомлений вибір, що підвищує конверсію та робить взаємодію з платформою більш прозорою та продуктивною.

Така стратегія розмежування прав доступу дозволяє збалансувати відкритість інформаційних ресурсів із необхідністю захисту комерційних інтересів проєкту (рис 3.24).

```
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. 3.24. Код шаблонів з водяними знаками для незареєстрованих користувачів

Програмна реалізація компонента TemplateList

Компонент **TemplateList** є ключовим елементом інтерфейсу, що відповідає за динамічне відображення переліку доступних шаблонів документів та надання інструментарію для управління ними. Архітектура компонента побудована на принципах модульності та інтеграції з глобальним станом додатка.

Функціональні особливості та структура компонента:

• Управління даними через Props та Redux. Компонент отримує вхідний параметр allTemplates — масив об'єктів, що містить метадані та посилання на

візуальні образи шаблонів. Взаємодія з базою даних Firebase здійснюється опосередковано через диспетчеризацію Redux-екшенів, що забезпечує синхронізацію інтерфейсу з актуальним станом системи.

• Механізм видалення ресурсів. При ініціації операції видалення активується функція handleDelete(id). Вона ініціює асинхронний запит до сховища, після чого відповідний шаблон вилучається як із бази даних, так і з поточного стану відображення без перевантаження сторінки.

• Управління персоналізованим списком («Обране»). Функція handleFavorite(templateId, bool) дозволяє користувачу маркувати пріоритетні шаблони. Логіка передбачає динамічне оновлення властивостей об'єкта в Redux-сторі, що миттєво змінює візуальний статус елемента (наприклад, стан іконки).

• Навігація та редагування. Кожен елемент списку містить інтерактивне посилання та кнопку переходу до редактора TinyMCE. При кліку на зображення шаблону система ініціює маршрутизацію до відповідної сторінки редагування, передаючи унікальний ідентифікатор (ID) обраного документа.

• Візуальна декомпозиція (TransportationItem). Для оптимізації рендерингу кожен шаблон інкапсульовано в дочірній компонент, що включає назву документа та його графічну прев'ю-копію. Це дозволяє реалізувати механізм "лінивого завантаження" (lazy loading) для великих списків.

• Стилізація та інтерфейс. Оформлення компонента реалізовано за допомогою бібліотеки Styled Components (файл TemplatesList.styled). Це забезпечує ізоляцію стилів для кнопок управління, ефектів наведення (hover) та адаптивну сітку для коректного відображення каталогу на пристроях з різною роздільною здатністю екрана.

Загалом, **TemplateList** виконує роль інтерактивного диспетчера, який поєднує візуальну презентацію шаблонів із функціональними можливостями CRUD-операцій (створення, читання, оновлення, видалення), забезпечуючи високу ергономіку користувацького інтерфейсу (рис 3.25).

```
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 (
  <Container>
    {allTemplates?.map(template => (
      <TransportationItem key={template.id}>
        <P>{template.name}</P>
        <RoutH>
          <Link
            to={`/template-edit/${template.id}`}
            style={{ height: '100%' }}
          >
            <Image src={template.templateImageUrl} alt={template.name} />
          </Link>
          <Rout>
            {' '}
            <EditButton
              to={`/template-edit/${template.id}`}
              title="Редагувати шаблон"    "Редагувати": Unknown word.
            >
              <RiEdit2Line size={25} />
            </EditButton>
            {template?.email !== user?.email && (
              <FavButton
                title="Додати до обраних"    "Додати": Unknown word.
                col={favTemplates?.findIndex(
                  temp => temp.uid === template.id
                )}
                onClick={() => {
                  favTemplates?.findIndex(temp => temp.uid === template.id)
                    ? handleFavorite(template.id, true)
                    : handleFavorite(
                        favTemplates?.find(temp => temp.uid === template.id),
                        false
                      );
                }} ← #62-69 onClick=
              >
                <RiHeartAddLine size={20} />
              </FavButton>
            )} ← #56-73 </EditButton>
            {template?.email !== user?.email && (
              <DeleteButton
                title="Видалити шаблон"    "Видалити": Unknown word.
                onClick={() => {
                  handleDelete(template.id);
                }}
              >
                <RiDeleteBinLine size={20} />
              </DeleteButton>
            )} ← #74-83 {template?.email !== user?.email && (
          </Rout>
        </RoutH>
      </TransportationItem>
    ))} ← #38-87 <Container>
  </Container>
); ← #36-89 return
```

Рис. 3.25. Код компонента TemplateList

Програмна реалізація модуля пошуку та фільтрації (TemplateFind)

Компонент TemplateFind забезпечує функціональність селективного відбору та пошуку шаблонів, що є критично важливим для оптимізації

взаємодії користувача з великими масивами даних. Модуль реалізовано як функціональний компонент React, що базується на принципах реактивного оновлення інтерфейсу.

Технічні особливості та функціональні елементи:

• Управління локальним станом. За допомогою хука useState реалізовано контроль стану searchTerm, який динамічно зберігає текстовий запит користувача. Це забезпечує актуалізацію даних у реальному часі під час введення.

• Інтерактивне введення (Input). Поле введення інтегроване з функцією-обробником handleInputChange. Кожна зміна символу в полі ініціює оновлення стану, що дозволяє системі миттєво реагувати на дії користувача.

• Механізм пошуку (handleSearchClick). При активації елемента управління «Пошук» викликається функція зворотного виклику onSearch. Вона передає накопичений термін пошуку до батьківського компонента або Redux-стору для фільтрації основного масиву шаблонів за ключовими словами.

• Фільтрація за пріоритетністю (handleFavoriteClick). Окремий функціональний блок onFavorite відповідає за перемикання режимів відображення. Він дозволяє користувачеві миттєво відфільтрувати список, залишаючи лише марковані («обрані») шаблони, що значно спрощує навігацію в межах персоналізованого контенту.

• Стилізація та архітектура інтерфейсу. Візуальне представлення модуля інкапсульовано у файлі TemplateFind.styled. Використання бібліотеки Styled Components дозволяє чітко розмежувати логіку пошуку та її стилістичне оформлення, забезпечуючи адаптивність контейнерів, кнопок та полів введення для різних типів пристроїв [17].

Завдяки впровадженню TemplateFind, інтерфейс стає динамічним та адаптивним, надаючи користувачеві інструменти для швидкої ідентифікації необхідних документів серед загальної бази шаблонів (рис 3.26).

```
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. 3.26. Код компонента TemplateFind

3.2. Тестування системи створення шаблонів.

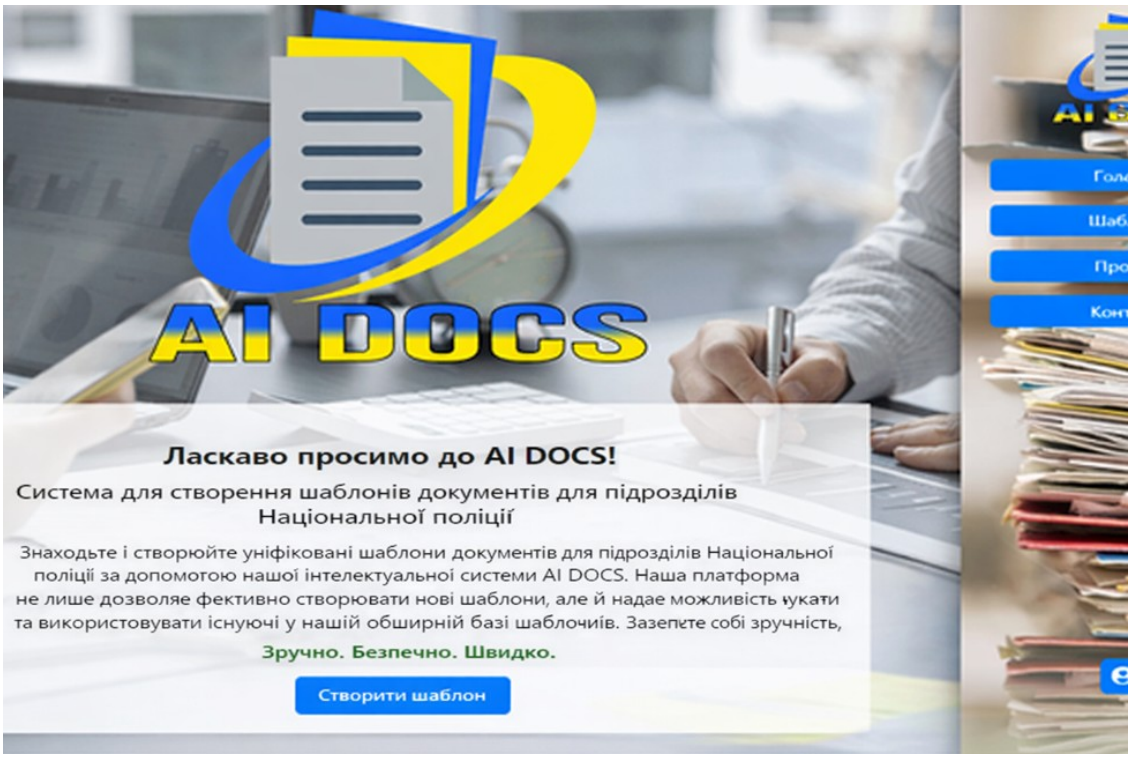


Рис. 27. Головна сторінка

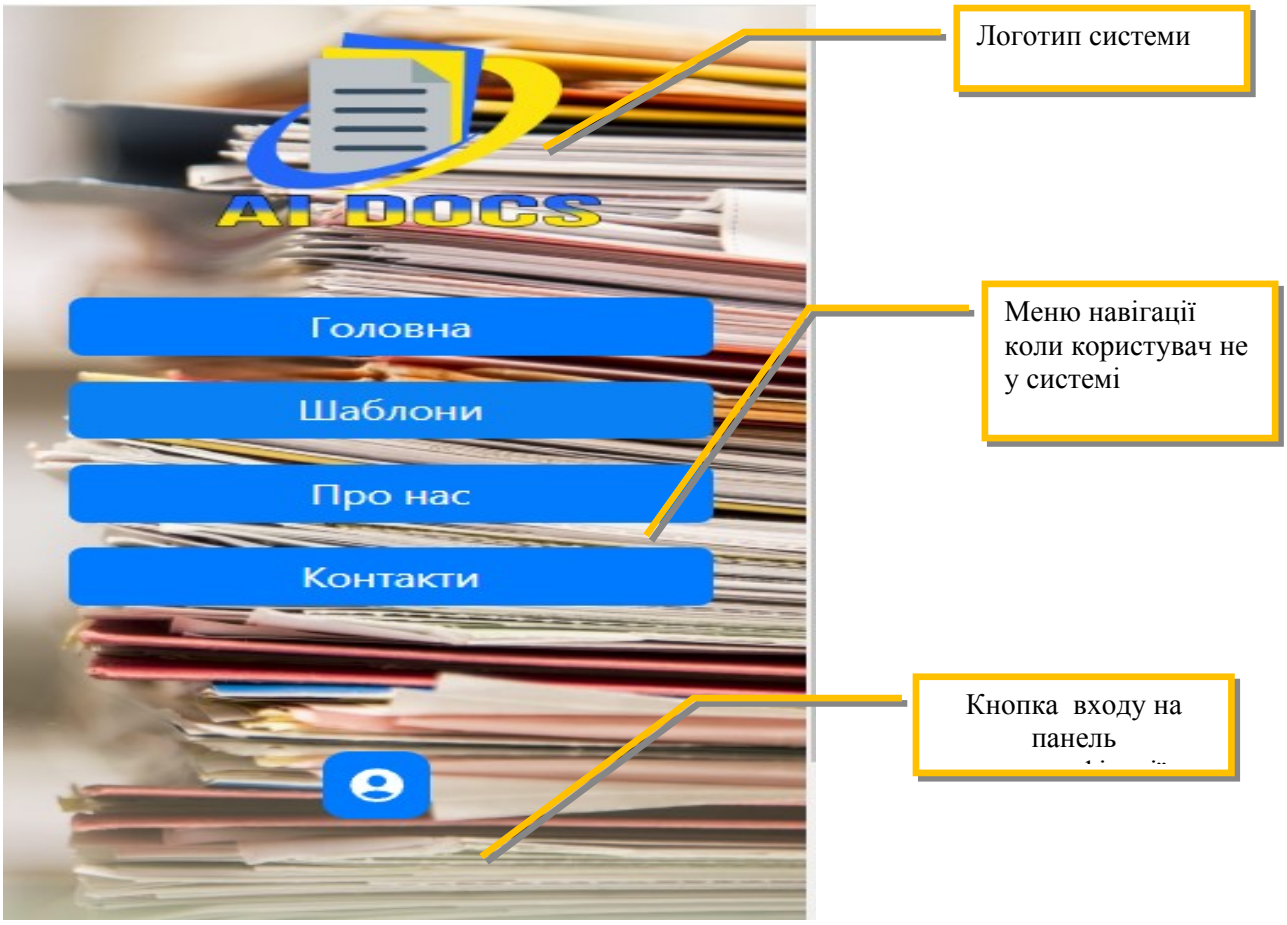


Рис. 28. Інтерфейс навігаційної панелі з інтегрованим модулем автентифікації.

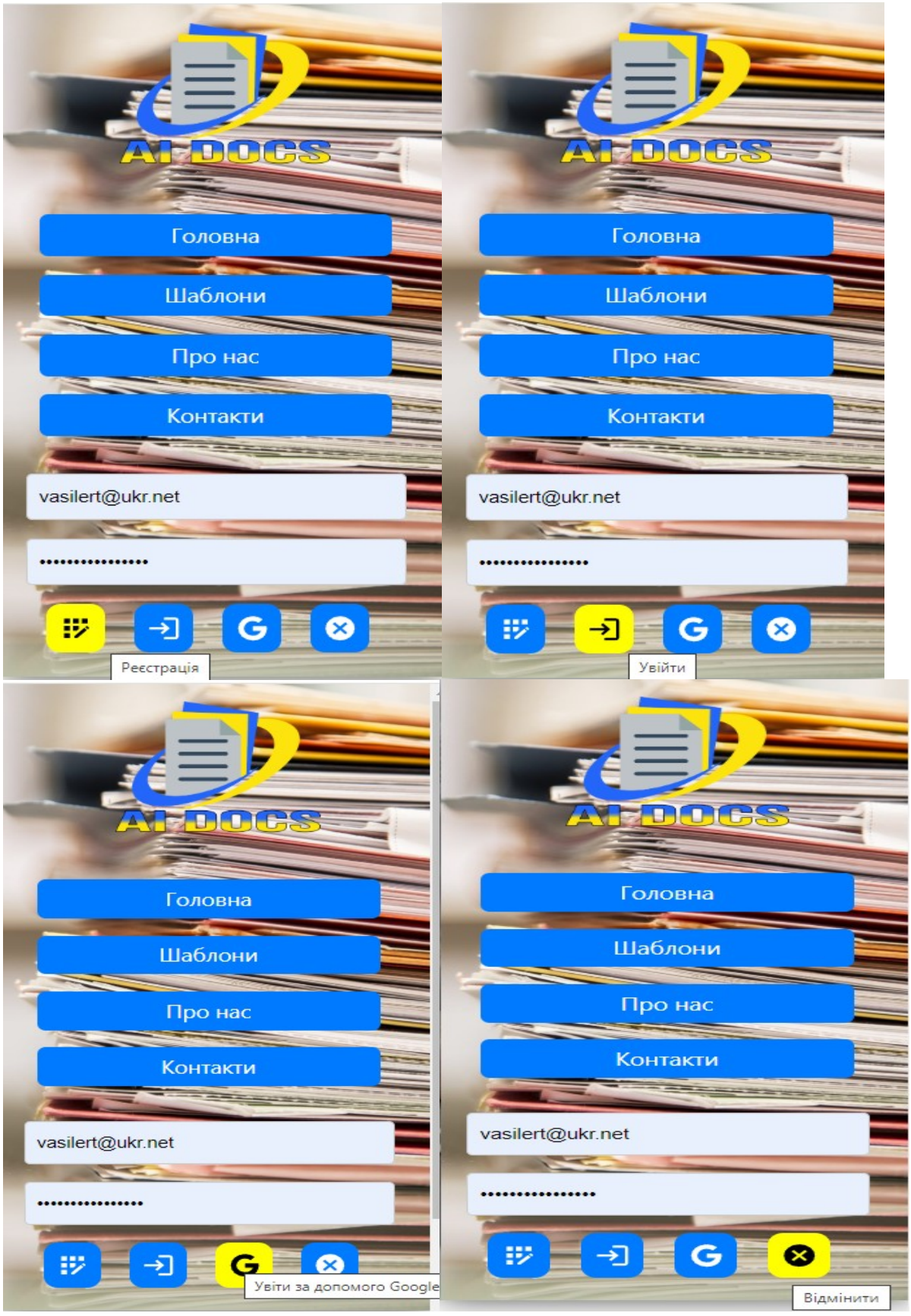


Рис. 29. Функціональні компоненти панелі входу та реєстрації.

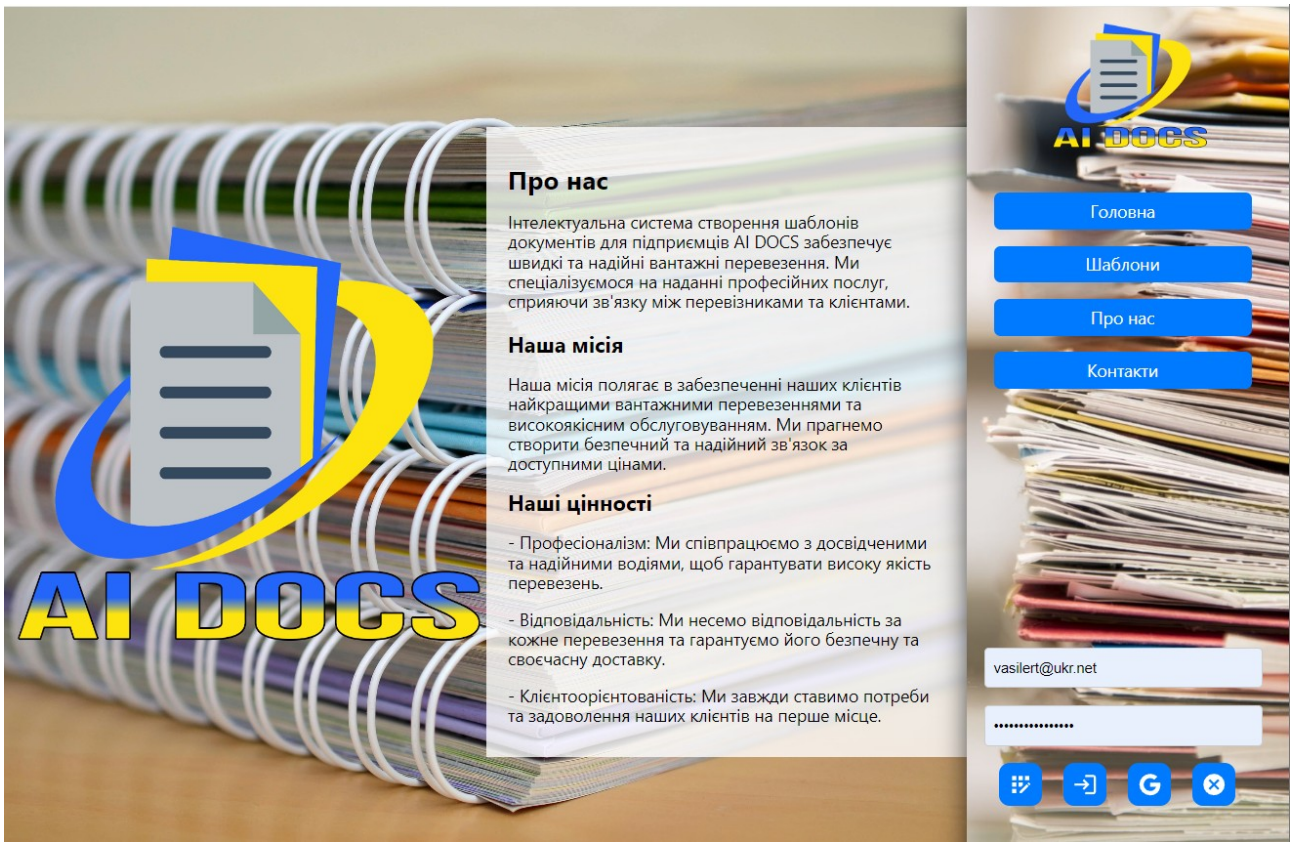


Рис. 30. Сторінка «Про нас».

Рис. 31. Сторінка «Контакти».

Рис. 32. Персоналізоване меню навігації з доступом до розширених функцій системи.

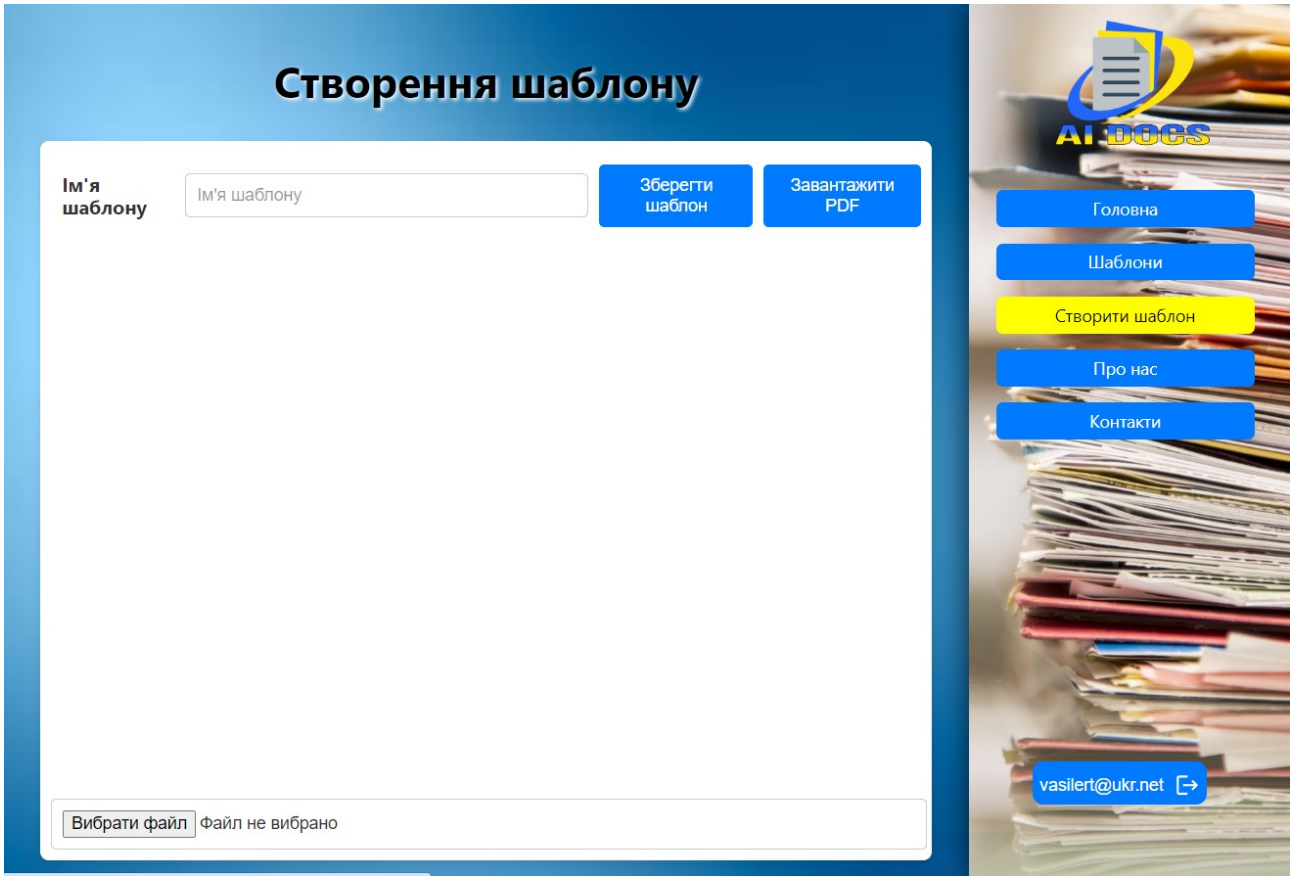


Рис. 33. Інтерфейс робочої області для створення нового шаблону

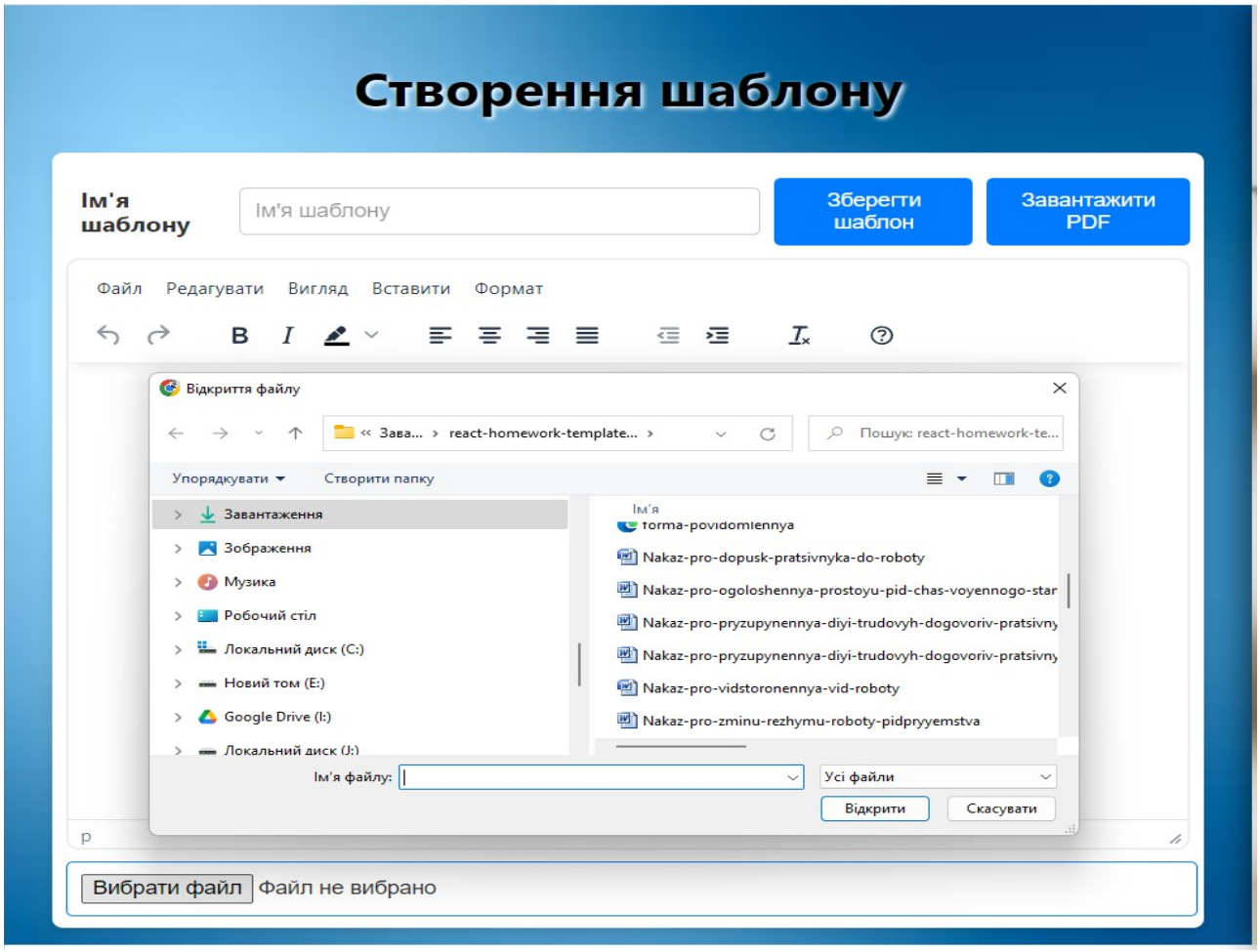


Рис. 34. Етап завантаження та підготовки шаблону користувача до модифікації.

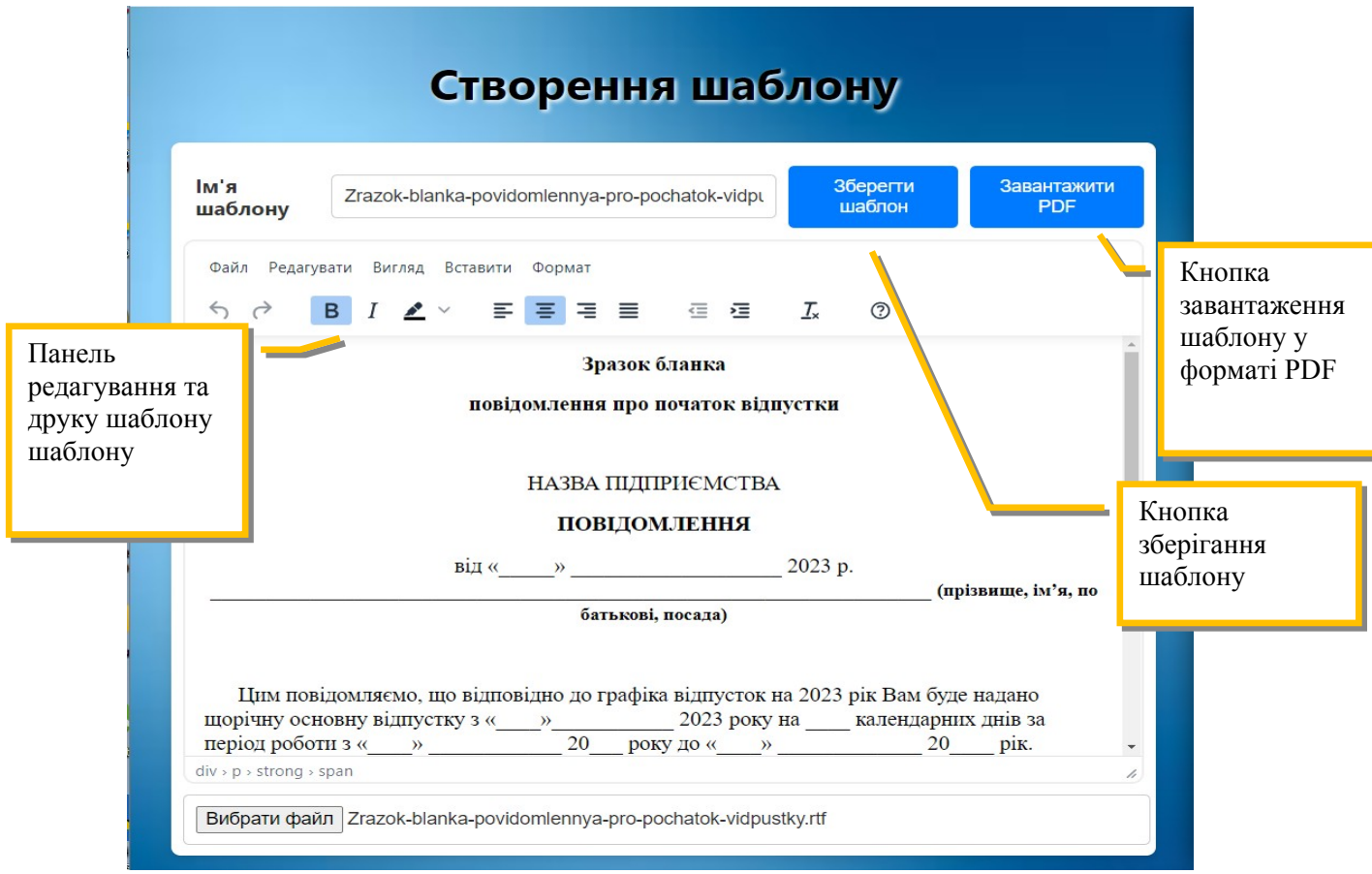


Рис. 35. Редагування та збереження шаблону

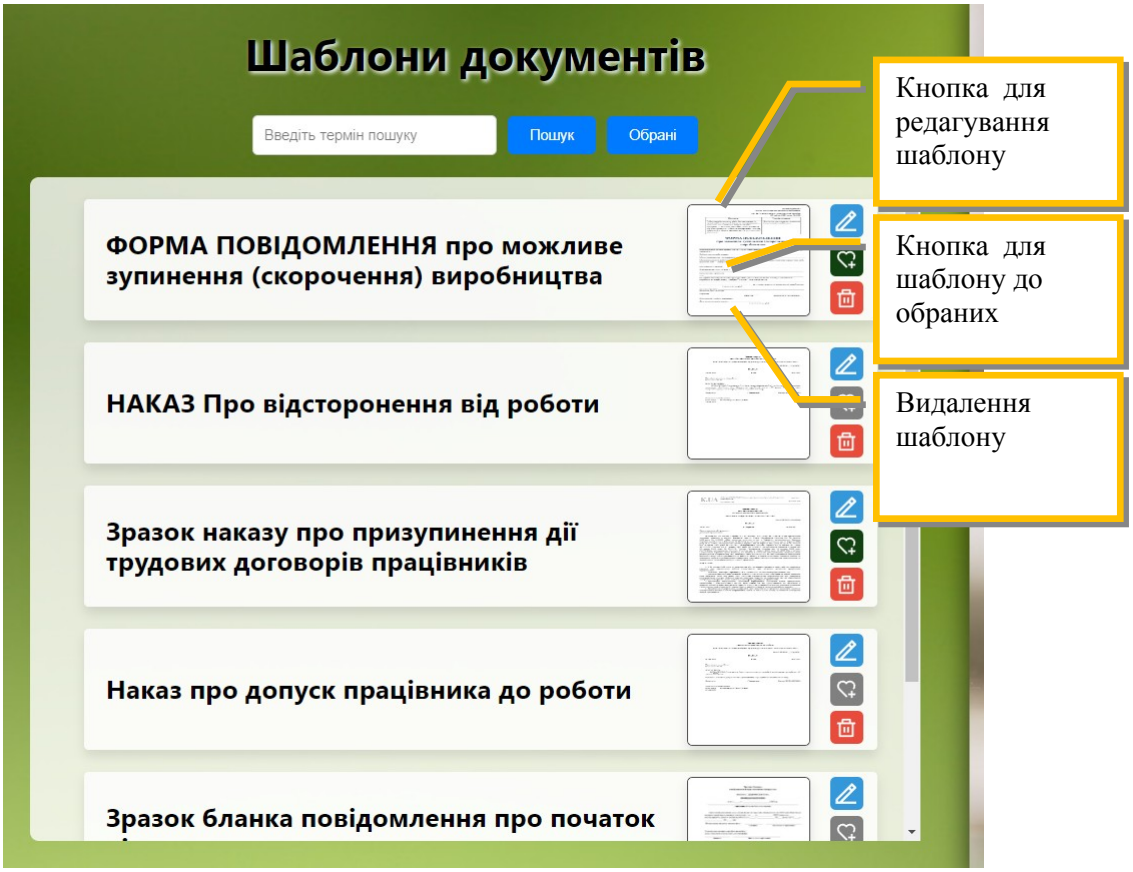


Рис. 36. Загальний вигляд сторінки перегляду та вибору шаблонів.

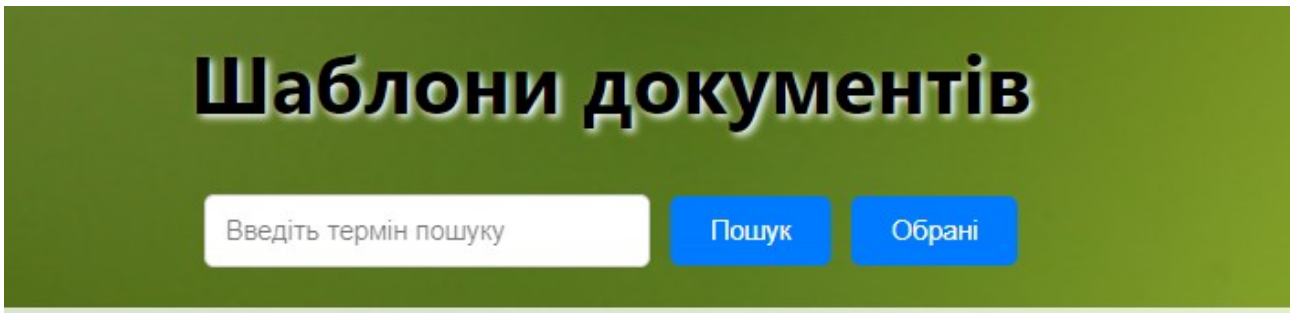


Рис. 37. Панель пошуку

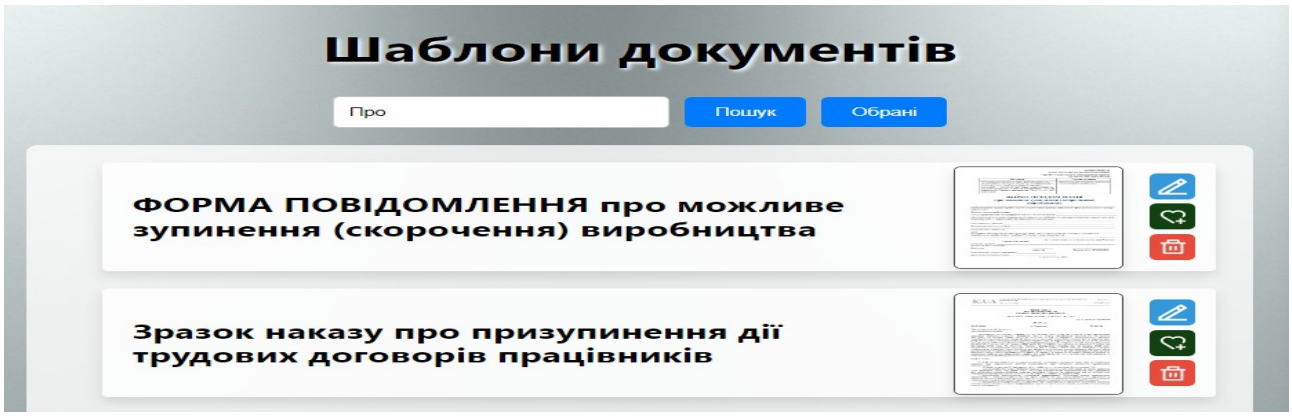


Рис. 38. Обрані шаблони

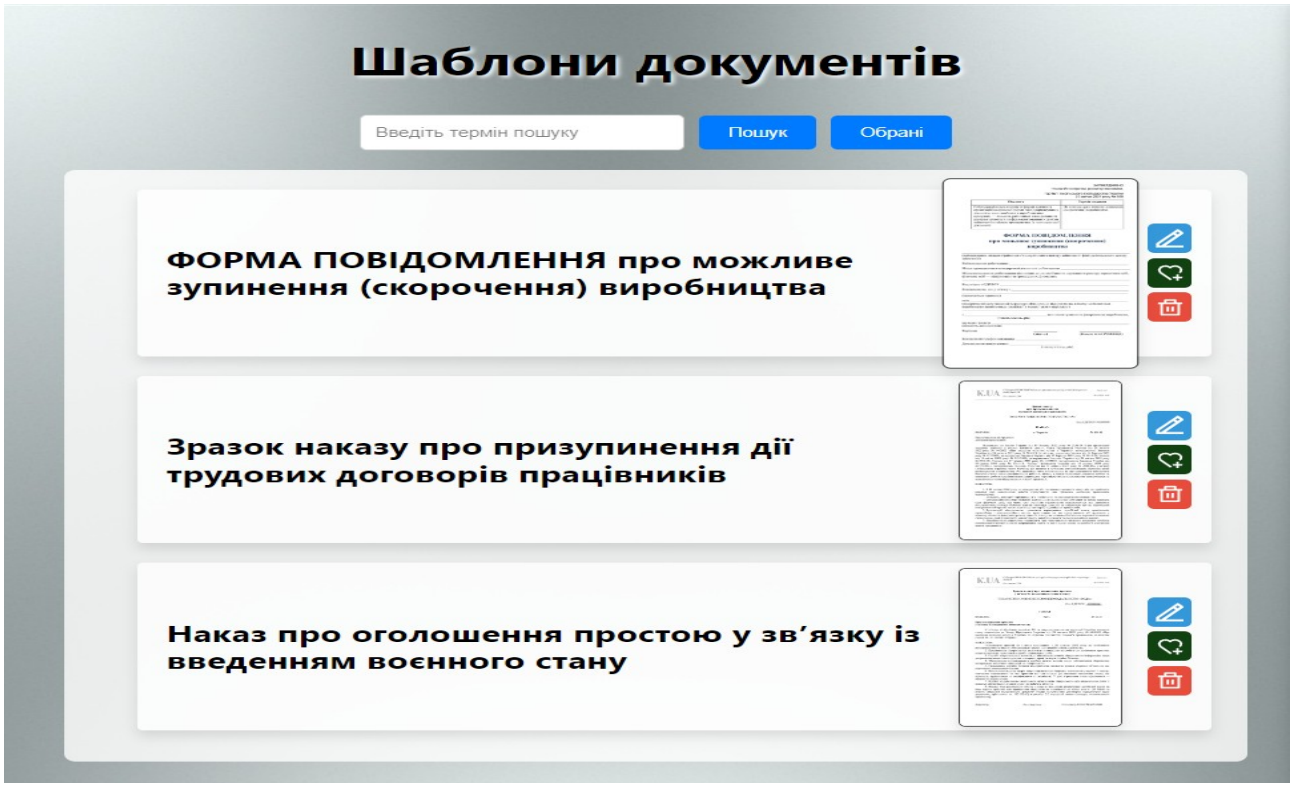


Рис. 39. Обрані шаблони користувача

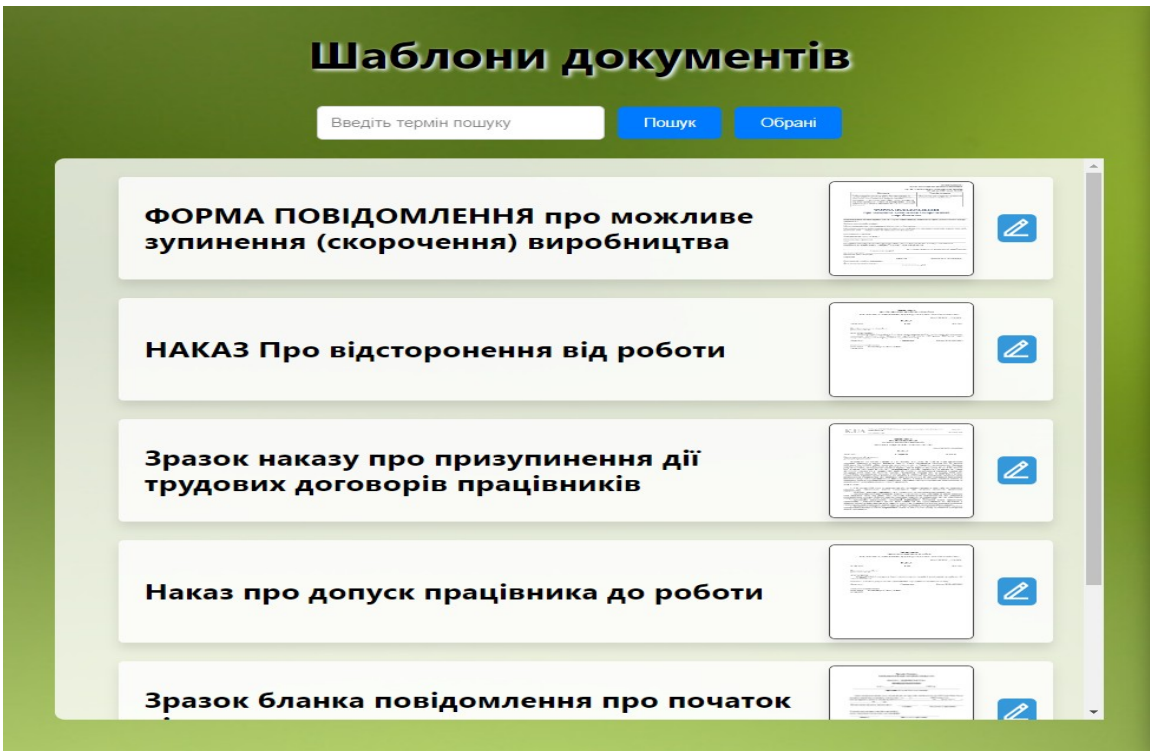


Рис. 40. Інтерфейс перегляду каталогу шаблонів для неавторизованих відвідувачів.

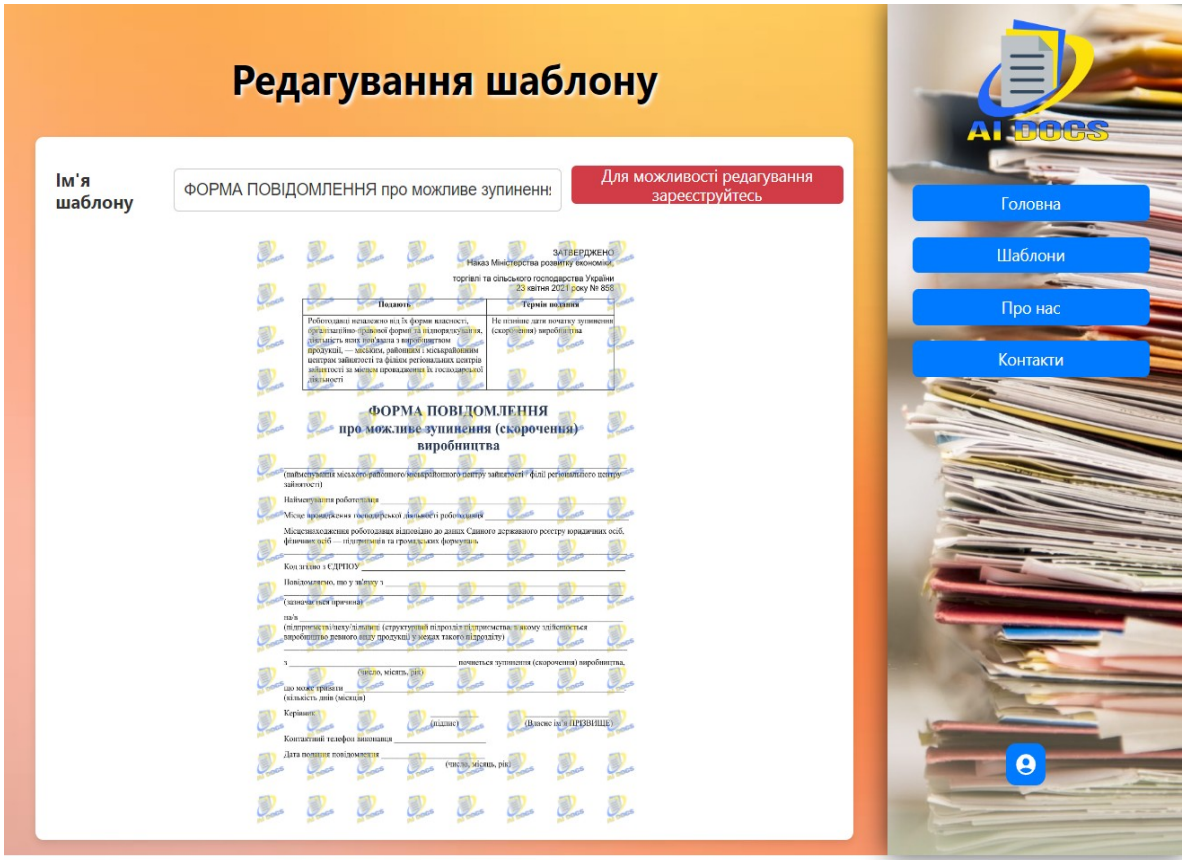


Рис. 41. Візуалізація робочої області редактора для неавторизованих користувачів.

Висновки до розділу

Підсумовуючи результати проєктування інтелектуальної системи AiDocs, слід зазначити, що використання сучасних методологій веб-розробки, зокрема бібліотеки React, стало фундаментом для створення інноваційного рішення в автоматизації ділового документообігу. Проведений аналіз дозволяє виділити такі ключові аспекти:

• Модульність та компонентний підхід. Використання React забезпечило можливість декомпозиції складного інтерфейсу на ізольовані, повторно використовувані компоненти. Це не лише пришвидшило процес архітектурної збірки платформи, а й гарантувало високу ремонтопридатність коду. Кожен функціональний блок — від редактора контенту до систем фільтрації — може бути незалежно модифікований без порушення цілісності всієї системи.

• Динамічне управління станом. Впровадження функціональних компонентів у поєднанні з хуками (useState, useEffect, useContext) дозволило реалізувати реактивну модель поведінки додатка. Система миттєво реагує на дії користувача (пошук у реальному часі, динамічна автентифікація, автозаповнення полів шаблону), що забезпечує високу швидкість відгуку інтерфейсу та мінімізує затримки при опрацюванні великих масивів даних.

• Масштабованість та декларативність. Декларативний підхід до опису інтерфейсів дозволив абстрагуватися від маніпуляцій з DOM-деревом, зосередивши увагу на бізнес-логіці управління документами. Це закладає надійну основу для подальшого розширення функціонала AiDocs, зокрема інтеграції складних алгоритмів аналізу тексту та розширених засобів спільної роботи над проєктами.

• Оптимізація користувацького досвіду (UX). Завдяки поєднанню архітектури React із сучасними інструментами стилізації та маршрутизації, вдалося створити безшовне робоче середовище. Односторінкова структура додатка (SPA) виключає необхідність повного перевантаження сторінок, що робить процес створення та редагування ділових паперів максимально наближеним до роботи з нативними десктопними програмами.

Таким чином, обрана стратегія розробки забезпечує високу конкурентоспроможність платформи AiDocs, поєднуючи технічну ефективність коду з ергономічністю та надійністю інструментів управління корпоративною документацією.

ВИСНОВКИ

Узагальнюючи результати проведеного дослідження та розробки, можна стверджувати, що мета дипломної роботи щодо проєктування та реалізації високоефективної системи керування документами AiDocs була повністю досягнута. Аналіз сучасних тенденцій цифровізації підтвердив стратегічну важливість впровадження систем електронного документообігу як ключового чинника модернізації та оптимізації діловодних процесів. Перехід до автоматизованого управління документацією сприяє стандартизації управлінської діяльності, забезпечує високу швидкість доступу до інформації та дозволяє мінімізувати вплив людського фактора під час виконання рутинних операцій, що є критично важливим для підвищення загальної продуктивності колективу в умовах динамічного бізнес-середовища.

У ході виконання роботи було успішно вирішено низку складних науково-технічних завдань. Проведений аналіз архітектурних особливостей існуючих рішень дозволив виокремити найбільш затребувані функціональні можливості, що лягли в основу концепції AiDocs. Фундаментальною особливістю розробленої системи став компонентний підхід, який дозволив структурувати програмний продукт у вигляді сукупності взаємопов’язаних, проте автономних модулів. Таке рішення забезпечило значне спрощення процесів розробки, налагодження та подальшої модифікації коду, створюючи сприятливі умови для оперативного впровадження нового функціоналу та адаптації платформи до змінних вимог ринку. Використання сучасного стеку технологій, зокрема бібліотеки React для побудови інтерфейсу та середовища Node.js для реалізації серверної логіки, забезпечило створення стабільної, масштабованої та відмовостійкої платформи.

Особливе значення має наукова новизна розробленого рішення, що полягає у впровадженні архітектури попередньо запрограмованого контролера управління станом. Це дозволило значно оптимізувати ініціалізацію робочого

середовища редактора та пришвидшити обробку запитів, забезпечуючи миттєву реакцію системи на дії користувача. Застосування віртуального DOM та механізмів реактивного оновлення даних дозволило досягти високої продуктивності графічного інтерфейсу навіть при роботі зі складними структурами документів, що містять велику кількість динамічних полів та елементів кастомізації.

Практичне значення роботи полягає у створенні готового до експлуатації програмного продукту з високим рівнем зручності та доступності інтерфейсу. Система дозволяє персоналу без спеціальних навичок програмування чи верстки ефективно завантажувати, редагувати та використовувати професійні шаблони, забезпечуючи високу якість фінальної документації. Впровадження модулів автентифікації та розмежування прав доступу забезпечило надійний захист даних, де авторизовані користувачі отримують доступ до хмарного сховища персоналізованих шаблонів, реалізованого на базі інфраструктури Firebase. Це гарантує цілісність інформації та миттєву синхронізацію станів між клієнтською та серверною частинами застосунку.

Важливим аспектом реалізації стала підсистема експорту, яка дозволяє здійснювати рендеринг та конвертацію сформованих матеріалів у загальноприйнятий формат PDF. Це забезпечує універсальність використання документів поза межами системи, підтримуючи процеси друку та офіційного обміну інформацією. Результати практичного тестування підтвердили, що впровадження AiDocs дозволяє скоротити час на підготовку типової звітності та договорів на 30–40% за рахунок автоматизації рутинних операцій та використання інтегрованої бази шаблонів.

Перспективи подальших досліджень та розвитку проєкту вбачаються у розширенні інтелектуальних можливостей платформи, зокрема шляхом впровадження алгоритмів машинного навчання для автоматичного аналізу контексту та надання рекомендацій щодо заповнення полів. Також розглядається можливість реалізації модулів для колективної роботи в

реальному часі та інтеграції із зовнішніми хмарними сервісами електронного цифрового підпису.

Підсумовуючи, можна стверджувати, що створена система має високу прикладну цінність, відповідає вимогам сучасної програмної інженерії та є перспективною базою для подальшої цифровізації корпоративного управління документацією.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React Documentation. Official website : вебсайт. URL: https://react.dev/ (дата звернення: 21.05.2026).
2. Barklund M.React in Depth.Shelter Island: Manning Publications,2024.450 p.
3. Barklund M., Mardan A. React Quickly. 2nd ed. Shelter Island : Manning Publications, 2023. 525 p.
4. Коротєєва Т. О. Технології створення програмних продуктів : навч. посібник. Львів : Видавництво Львівської політехніки, 2021. 240 с.
5. Мартин Р. Чиста архітектура. Мистецтво розробки програмного забезпечення. Харків : Фабула, 2019. 368 с.
6. Гівербек М. Виразний JavaScript. Сучасне програмування. Київ : Видавець Олександр Савчук, 2019. 480 с.
7. http://graphics.cs.ucdavis.edu : вебсайт (сайт з КГ інституту аналізу даних і візуалізації Каліфорнійського університету) (дата звернення: 21.05.2026).
8. http://www.cg.tuwien.ac.at/courses/cg2 : вебсайт (сайт Інституту комп’ютерної графіки і алгоритмів Віденського технічного університету) (дата звернення: 21.05.2026).
9. Redux Toolkit. Modern Redux with RTK Query : вебсайт. URL: https://redux-toolkit.js.org/ (дата звернення: 21.05.2026).
10. Node.js v25.8.1 Documentation. API reference : вебсайт. URL: https://nodejs.org/api/ (дата звернення: 21.05.2026).
11. Firebase Documentation. Cloud Firestore & Authentication : вебсайт. URL: https://firebase.google.com/docs/ (дата звернення: 21.05.2026).
12. Wieruch R. The Road to React: Your journey to master plain yet pragmatic React. Berlin : Leanpub, 2023. 364 p.
13. TinyMCE Documentation. Content formatting options : вебсайт. URL: https://www.tiny.cloud/docs/ (дата звернення: 21.05.2026).

14. Svelte Documentation. Svelte 5 and beyond : вебсайт. URL: https://svelte.dev/docs (дата звернення: 21.05.2026).
15. Zandstra M. PHP Objects, Patterns, and Practice. 6th ed. New York : Apress, 2021. 598 p.
16. Walker J. R., Dixon B. Machining Fundamentals. 10th ed. Goodheart-Willcox, 2018. 642 p.
17. Wilson B. A. GD&T: Application and Interpretation. 7th ed. Goodheart-Willcox, 2019. 512 p.

ДОДАТКИ

Додаток А

Програмний код для реалізування шаблонів документів для підрозділів Національної поліції

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": Unknowr
    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>    "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%',
```

```
                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»

```
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»

```
ges > Templates > templates.jsx > Templates
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;
```