Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка веб-сайту для систематизації та представлення сучасних інструментів на основі штучного інтелекту

Виконав: студент IV курсу, групи СН-42
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Мельник Р.Р.
(підпис) (прізвище та ініціали)

Керівник _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ _____
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«_8_» _червня_ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)
за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)
Студенту Мельнику Роману Руслановичу
(прізвище, ім'я, по батькові)
1. Тема роботи Розробка веб-сайту для систематизації та представлення сучасних інструментів на основі штучного інтелекту

Керівник роботи Боднарчук І.О., к.т.н., доцент, завідувач кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «*14*» *травня* 2026 року № *4/9-239*
2. Термін подання студентом завершеної роботи 22 червня 2026 р.
3. Вихідні дані до роботи Вхідні дані включають технічні специфікації сервісів ші, категорії моделей, масив відгуків фахівців, query-параметри фільтрів. Вихідні дані містять бд NoSQL, SPA веб-сайт, картки сервісів та автономну логіку розділу обраного на базі LocalStorage.
4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз предметної області та постановка завдання. 1.1 Тенденції розвитку сучасного ринку моделей штучного інтелекту. 1.2 Критичний аналіз існуючих каталогів-аналогів. 1.3 Функціональні вимоги до вебресурсу. 1.4 Висновок до першого розділу. 2. Проектна частина та обґрунтування методів розробки. 2.1 Обґрунтування технологічного стеку MERN та інструментів збірки. 2.2 Проектування структури бази даних MongoDB. 2.3 Специфікація полів та типізація метаданих. 2.4 Реалізація REST API. 2.5 Логіка збереження даних у LocalStorage. 2.6 Висновок до другого розділу. 3. Програмна реалізація та тестування інформаційної системи. 3.1 Налаштування середовища розробки. 3.2 Реалізація серверної логіки Express.js. 3.3 Розробка клієнтського інтерфейсу на React. 3.4 Тестування та верифікація функціоналу. 3.5 Висновок до третього розділу. 4. Безпека життєдіяльності, основи охорони праці. Висновки. Перелік джерел. Додатки.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Титульний слайд. 2. Актуальність розробки. 3. Мета і завдання проєкту. 4. Об'єкт та предмет дослідження. 5. Розбір недоліків аналогів. 6. Вибір стеку MERN. 7. Відмова від авторизації на користь LocalStorage. 8. Структура документа бази даних MongoDB. 9. Реалізація серверної частини. 10. Сторінка каталогу та картки ШІ. 11. Сторінка обраного. 12. Адаптація інтерфейсу на мобільних пристроях. 13. Загальні висновки. 14. Дякую за увагу!

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Мариненко С.Ю, к.т.н., доцент каф. МТ	01.06.2026	08.06.2026

7. Дата видачі завдання 26 січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01.2026	*Виконано*
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	27.01.2026-16.02.2026	*Виконано*
3.	Виконання дослідження щодо процесів категоріального розподілу, систематизації та інтерактивного представлення прикладних інструментів на основі штучного інтелекту.	17.02.2026-10.05.2026	*Виконано*
	Розроблення трирівневої клієнт-серверної архітектури, структури нереляційної бази даних MongoDB, асинхронного Express REST API та адаптивного React-інтерфейсу системи.		
4.	Оформлення розділу «Аналіз предметної області та постановка завдання»	11.05.2026-17.05.2026	*Виконано*
	»		
5.	Оформлення розділу «Проектна частина та обґрунтування методів дослідження»	18.05.2026-24.05.2026	*Виконано*
	»		
6.	Оформлення розділу «Програмна реалізація та експериментальне тестування веб-сайту для представлення сучасних інструментів штучного інтелекту»	25.05.2026-31.05.2026	*Виконано*
	»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	01.06.2026-08.06.2026	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	01.06.2026-08.06.2026	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	09.06.2026-11.06.2026	*Виконано*
10.	Нормоконтроль	12.06.2026-15.06.2026	*Виконано*
11.	Перевірка на плагіат	16.06.2026	*Виконано*
12.	Попередній захист кваліфікаційної роботи	18.06.2026	*Виконано*
13.	Захист кваліфікаційної роботи	24.06.2026	

Студент _____ Мельник Р.Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка веб-сайту для систематизації та представлення сучасних інструментів на основі штучного інтелекту // Кваліфікаційна робота освітнього ступеня «Бакалавр» // Мельник Роман Русланович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-42 // Тернопіль, 2026 // С. 56, рис. – 7, табл. – 2, кресл. – 14, додат. – 2, бібліогр. – 44.

Ключові слова: react, node.js, express, mongodb, веб-сайт, каталог, штучний інтелект, фільтрація, localstorage, vercel, render.

У цій бакалаврській роботі створено веб-сайт-каталог, де збираються та структуруються картки з інформацією про сучасні нейромережі та ШІ-продукти. Головна мета проєкту - зробити зручний пошук та порівняння сервісів штучного інтелекту, оскільки зараз їх стає занадто багато, а існуючі платформи часто перевантажені зайвими елементами.

Для реалізації обрано стек MERN. На бекенді працює Node.js разом із фреймворком Express.js. Вони забезпечують роботу REST API, яке обробляє запити користувача та дістає дані з хмарної бази MongoDB Atlas. Документи в базі даних спроектовані під параметри нейромереж: там зберігаються не просто назви й описи, а й технічні деталі - розмір контекстного вікна, тип ліцензії, наявність мобільного додатка, API і так далі. Робота зі сторонніми API відсутня, цей сайт не є агрегатором, вся інформація є статичною та завантажується суто з бази даних додатка.

Фронтенд сайту побудовано на React за допомогою інструменту збірки Vite. Щоб каталог миттєво реагував на вибір фільтрів та пагінацію, підключено менеджер станів Redux Toolkit. Важлива фішка інтерфейсу - повна відмова від реєстрацій та логінів. Список "Обраного" повністю працює через LocalStorage в браузері. Це максимально спрощує роботу з сайтом, не змушує користувача залишати свої дані та пришвидшує взаємодію з платформою. Готовий код у

вигляді монорепозиторію завантажено на GitHub, бекенд задеплоєно на Render.com, а клієнтську частину (фронтенд) - на хостингу Vercel.

Об'єкт дослідження: процес організації, збереження та відображення інформації про ШІ-сервіси у веб-середовищі.

Предмет дослідження: використання інструментів React, Node.js, маршрутизації Express та бази даних MongoDB для створення адаптивного сайту-каталогу.

ANNOTATION

Development of a Website for Systematizing and Presenting Modern AI-Based Tools // Qualification work of the educational level «Bachelor» // Melnyk Roman Ruslanovych // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group SN-42 // Ternopil, 2026 // P. 56, fig. – 7, tabl. – 2, chair. – 14, annexes. – 2, references – 44.

Keywords: react, node.js, express, mongodb, website, catalog, artificial intelligence, filtering, localstorage, vercel, render.

In this bachelor thesis, a website catalog has been developed to collect and structure information cards about modern neural networks and AI products. The main goal of the project is to provide a convenient search and comparison of artificial intelligence services, as they are becoming too numerous now, while existing platforms are often overloaded with unnecessary elements.

The MERN stack was chosen for implementation. Node.js works on the backend together with the Express.js framework. They provide the operation of the REST API, which processes user requests and retrieves data from the cloud database MongoDB Atlas. Documents in the database are designed specifically for neural network parameters: they store not just names and descriptions, but also technical details - context window size, license type, availability of a mobile application, API, and so on. There is no integration with third-party APIs, this website is not an aggregator, and all information is static and loaded purely from the application database.

The frontend of the website is built on React using the Vite build tool. The Redux Toolkit state manager is connected so that the catalog instantly responds to filter selection and pagination. An important feature of the interface is the complete rejection of registration and logins. The "Favorites" list works completely through

LocalStorage in the browser. This maximally simplifies interaction with the website, does not force users to leave their data, and speeds up interaction with the platform. The finished code is uploaded to GitHub as a monorepository, while the backend is deployed separately on Render.com, and the client side (frontend) is hosted on Vercel.

Object of research: the process of organizing, storing, and displaying information about AI services in a web environment.

Subject of research: using React, Node.js, Express routing, and MongoDB database tools to create an adaptive catalog website.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) - інтерфейс прикладного програмування.

JSON (JavaScript Object Notation) - текстовий формат обміну даними, що базується на парах ключ-значення.

MERN (MongoDB, Express, React, Node) - стек технологій для розробки веб-сайтів на базі мови JavaScript.

REST (Representational State Transfer) - архітектурний стиль взаємодії компонентів веб-додатка за допомогою HTTP-запитів.

SPA (Single Page Application) - односторінковий веб-додаток, що оновлює контент без повного перезавантаження сторінки в браузері.

UI (User Interface) - користувацький інтерфейс, графічне оформлення та елементи керування сайту.

UX (User Experience) - досвід користувача, що визначає зручність та логіку взаємодії з інтерфейсом.

БД - база даних.

СУБД - система керування базами даних.

ШІ - штучний інтелект.

ЗМІСТ

ВСТУП

Актуальність теми. Сьогодні сфера генеративного штучного інтелекту розвивається дуже швидко, через що щодня з'являються десятки нових нейромереж під різні завдання: від написання коду до генерації аудіо чи обробки зображень. Проте звичайним користувачам та розробникам стає дедалі важче орієнтуватися у цьому потоці, оскільки існуючі агрегатори часто перевантажені рекламою, складною навігацією або вимагають обов'язкової реєстрації навіть для швидкого перегляду характеристик. Створення простого, швидкого та адаптивного веб-сайту для систематизації інструментів ШІ дозволяє структурувати актуальні сервіси за чіткими параметрами, спрощуючи процес пошуку та вибору потрібного технологічного рішення.

Мета і задачі дослідження. Метою кваліфікаційної роботи є проектування та програмна реалізація адаптивного веб-сайту для систематизації, збереження та представлення інформації про сучасні інструменти штучного інтелекту на основі стеку технологій MERN.

Для досягнення поставленої мети потрібно виконати ряд завдань, зокрема:

– Проаналізувати існуючі веб-платформи для пошуку нейромереж, виявити їхні недоліки та сформувати вимоги до майбутнього сайту-каталогу.

– Розробити структуру бази даних MongoDB для гнучкого збереження технічних параметрів та характеристик ШІ-інструментів.

– Реалізувати серверну частину додатка на базі Node.js та Express.js для створення REST API та віддачі даних із бази.

– Розробити візуальний інтерфейс користувача на React із використанням збирача Vite та налаштувати логіку миттєвої фільтрації та пагінації карток.

– Інтегрувати менеджер станів Redux Toolkit та реалізувати роботу розділу "Обране" за допомогою технології LocalStorage без використання авторизації.

– Виконати деплой проекту, розгорнувши серверний код на платформі Render, а клієнтську частину - на хостингу Vercel.

Практичне значення одержаних результатів. Результатом роботи є готовий до використання веб-сайт, який виконує роль зручного довідника з архітектури та можливостей сучасного ШІ. Завдяки оптимізованому UX без обов'язкових логінів, користувачі можуть миттєво знаходити інструменти за категоріями, переглядати технічні деталі (наприклад ліцензії чи розмір контекстного вікна) та формувати власний список корисних сервісів. Створену систему можна використовувати як базу для тематичних IT-ресурсів або як внутрішній інструмент в компаніях для швидкого підбору безкоштовних чи комерційних нейромереж під робочі завдання.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Дослідження сучасного стану сфери штучного інтелекту та систем структуризації інструментів

За останні роки сфера штучного інтелекту переживає справжній технологічний бум, який повністю змінює правила гри на глобальному ринку праці та автоматизації процесів. Нові нейромережі, сервіси та готові ШІ-продукти з’являються в інтернет-просторі буквально щодня, і вони кардинально перевертають те, як люди звикли працювати з текстами, програмним кодом, графікою чи відеоконтентом. Зараз під будь-яку робочу задачу, від написання короткої відповіді клієнту до проектування великої архітектури коду, можна знайти готове рішення на базі великих мовних моделей або генераторів зображень. Компанії та незалежні розробники намагаються масово впроваджувати ці технології у свої повсякденні процеси, щоб максимально оптимізувати рутину та економити гроші. Проте такий шалений темп розвитку створює велику проблему інформаційного хаосу, в якому звичайній людині, програмісту чи менеджеру проекту стає дуже важко зорієнтуватися та зробити правильний вибір серед тисяч варіантів [13].

Головна складність сьогодні полягає в тому, що вся корисна інформація про нові моделі штучного інтелекту розкидана по всьому інтернету без будь-якого єдиного стандарту. Якісь цікаві релізи та анонси публікуються в соціальних мережах, щось активно обговорюється на закритих тематичних форумах розробників, а реальні технічні деталі взагалі доводиться вишукувати на офіційних блогах компаній-творців чи в інструкціях, які часто написані заплутано і сухо. Звичайному користувачу доводиться відкривати десятки вкладок у браузері, порівнювати описи з різних сайтів та самостійно перевіряти, чи не застаріла інформація, адже оновлення моделей відбуваються майже щотижня. Така децентралізація знань призводить до того, що етап

простого підбору потрібного інструменту перетворюється на повноцінне тривале дослідження, яке забирає ресурси, що могли бути спрямовані на реалізацію самого проекту.

Крім того, кожна окрема модель штучного інтелекту має купу власних унікальних параметрів, які критично важливо знати і розуміти ще перед початком роботи чи інтеграції. Наприклад, для текстових моделей та великих мовних систем життєво необхідно розуміти точний розмір контекстного вікна, адже це визначає, який об’єм тексту нейромережа здатна тримати в пам’яті за один раз без втрати логіки тривалого діалогу. Якщо обрати модель із замалим вікном контексту для аналізу великого юридичного договору чи довгого коду, система просто почне забувати початок документа, що призведе до критичних помилок. Для програмістів та системних архітекторів на перший план виходить тип ліцензії, тобто чи можна використовувати модель у комерційних цілях без юридичних ризиків, а також наявність стабільного API для швидкої інтеграції функцій ШІ у власний програмний продукт.

У той же час для дизайнерів, копірайтерів, маркетологів чи контент-мейкерів важливі зовсім інші практичні речі, які часто ігноруються в сухих технічних специфікаціях. Їм потрібна інформація про наявність офіційного мобільного додатка для швидкого доступу з телефона, підтримка голосових команд для зручного введення або вбудована технологія комп’ютерного зору (Vision), яка вміє розпізнавати та аналізувати завантажені картинки чи фотографії. Самостійний пошук, збір та ручна перевірка всіх цих різнопланових метаданих забирає занадто багато дорогоцінного часу, який фахівці змушені витрачати на рутину замість креативної діяльності. Без чіткої структури полів користувачі часто обирають розпіарені інструменти, які насправді не підходять під їхні технічні вимоги, просто через брак повної інформації.

Саме тому зараз в IT-індустрії виникає гостра і реальна потреба у створенні простих, зручних та прозорих сайтів для систематизації таких інструментів. Веб-сайти, які виконують роль каталогів, допомагають зібрати розрізнені дані в одному місці, верифікувати їх та розкласти по поличках за

чіткими і зрозумілими критеріями. Наявність такого ресурсу дозволяє структурувати ринок програмного забезпечення та дає користувачам можливість об’єктивно порівнювати різні моделі між собою. Це допомагає значно знизити поріг входження для новачків, які хочуть використовувати штучний інтелект у своїй роботі, але губляться у складній термінології та не знають, з чого почати пошук оптимального рішення.

Проте основна проблема більшості існуючих каталогів в інтернеті полягає в тому, що вони майже одразу після створення перетворюються на звичайні рекламні майданчики, де втрачається першочергова користь для відвідувача. Вони стають занадто перевантаженими з купою непотрібних кнопок та нав’язливими пропозиціями купити платну підписку. Власники таких ресурсів часто женуться за кількістю, додаючи тисячі сирих посилань. Користувач знову опиняється у ситуації, коли йому доводиться самостійно перебирати сотні карток у пошуках дійсно корисного інструменту, що нівелює саму ідею створення зручного каталогу.

Через це розробка легкого, чистого та швидкого веб-сайту з прозорою структурою даних є вкрай актуальною інженерною та практичною задачею. Створення простого каталогу, який фокусується не на кількості, а на якості та чіткій структуризації параметрів, дозволяє вирішити проблему швидкого підбору технологій. Такий інструмент допоможе користувачам буквально за два-три кліки миттєво знаходити потрібні сервіси під свої конкретні вимоги, не відволікаючись на рекламу, не вивчаючи складні інструкції та не виконуючи жодних зайвих реєстраційних дій у системі. Це робить розробку практично значущою для широкого кола спеціалістів і закладає надійну основу для подальшого аналізу та проектування архітектури всього додатку.

1.2 Порівняльний аналіз існуючих веб-ресурсів та аналогів

Для розуміння того, як краще побудувати власний каталог ШІ-сервісів, необхідно детально проаналізувати існуючі на світовому ринку рішення.

Сьогодні функціонує кілька великих міжнародних платформ, які професійно займаються збором, модерацією та сортуванням інструментів на базі штучного інтелекту. Найбільш відомими та відвідуваними серед них є сервіси під назвою There's An AI For That та Futurepedia. Поглиблений аналіз їхнього базового функціоналу, внутрішньої структури полів та користувацького інтерфейсу дозволить чітко виділити успішні інженерні рішення, а також знайти слабкі місця для оптимізації у власному проекті.

Перший детальний аналог, який варто розглянути, це міжнародна платформа There's An AI For That. Це один з найперших і найбільших каталогів у світі, який містить величезну базу нейромереж, що оновлюється в автоматичному та ручному режимах практично щодня. Головна особливість цього сайту полягає у веденні хронологічного таймлайну появи інструментів на ринку та використанні штучного інтелекту для пошуку всередині самого каталогу. Користувач може ввести свій запит звичайною розмовною мовою, і система намагається самостійно підібрати список релевантних сервісів. Проте гонитва за величезним об’ємом бази даних створює суттєвий мінус, адже інтерфейс сайту став занадто перевантаженим для сприйняття. На головній сторінці одночасно відображається дуже багато дрібних текстових елементів, рекламних блоків, категорій та нескінченних списків, через що звичайному користувачу важко швидко зорієнтуватися та виділити важливі технічні параметри конкретної моделі [36].

Другий популярний аналог, що заслуговує на увагу, це сервіс Futurepedia. Цей каталог має значно сучасніший, збалансований та чистий візуальний дизайн інтерфейсу. Інструменти тут чітко розділені за великими категоріями, реалізовано можливість перегляду популярних новинок за поточний тиждень або місяць, а також інтегровано систему збереження карток у закладки. Картки товарів виглядають структуровано і візуально привабливо, проте детальна технічна інформація про моделі часто повністю відсутня або схована глибоко в інтерфейсі сайту, що змушує виконувати багато зайвих кліків. Наприклад, розмір контекстного вікна мовної моделі чи специфічні особливості

ліцензування коду звичайний відвідувач просто не побачить на головній сторінці. Також для використання функцій персоналізації, таких як створення власних списків або підписка на оновлення, сайт вимагає від користувача обов’язкового створення облікового запису через пошту, що сильно уповільнює взаємодію з платформою, якщо потрібно просто швидко знайти точні параметри [37].

Для детального порівняння характеристик розглянутих аналогів та чіткого визначення подальшого вектору розвитку власного сайту-каталогу було сформовано таблицю 1.1.

Таблиця 1.1 - Порівняльний аналіз існуючих каталогів ШІ-інструментів

Критерій порівняння	**There's An AI For That**	**Futurepedia**	**Розроблюваний веб-сайт**
Об'єм бази даних	Дуже великий	Великий	Оптимальний
Складність інтерфейсу	Висока, перевантажена	Середня	Низька, мінімалістична
Наявність реклами	Так	Є спонсорські картки	Відсутня
Обов'язкова реєстрація	Необхідна для списків	Необхідна для списків	Відсутня
Технічні деталі ШІ (контекст, ліцензія)	Поверхневі дані	Переважно загальний опис	Повний набір структурованих полів
Швидкість оновлення інтерфейсу	Перезавантаження сторінок	Часткове оновлення	Миттєве (React Single Page Application)

Таким чином, проведений аналіз показує, що існуючі великі платформи орієнтовані на масштабність та монетизацію через рекламу, через що втрачається простота та швидкість взаємодії. Власний проект має на меті об'єднати найкращі сторони аналогів: надати користувачу чистий інтерфейс,

забезпечити миттєву роботу фільтрів без перезавантажень за рахунок React (див. додаток А, лістинг А.1) та дати можливість зберігати обране без реєстрації за допомогою LocalStorage.

1.3 Формування технічних вимог до платформи-каталогу

На основі аналізу предметної області та вивчення існуючих аналогів можна чітко сформулювати вимоги до створюваного веб-сайту. Головне завдання проєкту полягає у розробці легкого інформаційного ресурсу, який дозволить користувачам без зайвих перешкод і за мінімальну кількість кліків знаходити технічні дані про ШІ-сервіси. Відповідно до цього всі вимоги до системи розділяються на функціональні, які описують можливості користувача, та нефункціональні, які визначають технічні обмеження та параметри додатка.

Функціональні вимоги до веб-сайту включають декілька основних блоків, які повністю описують логіку взаємодії відвідувача з інтерфейсом додатка. Користувач повинен мати можливість без проблем відкривати головну сторінку, яка коротко ознайомлює його з призначенням платформи та містить помітну кнопку для швидкого переходу до каталогу. У самому каталозі має бути реалізовано зрозуміле виведення карток ШІ-інструментів, де відображається вся необхідна базова інформація, така як назва сервісу, розробник, рейтинг, короткий опис та іконка. Обов’язковою є наявність бічної панелі фільтрації, яка дозволить сортувати інструменти за категоріями, а також обирати специфічні параметри, наприклад, наявність мобільного додатка, API, підтримку технологій розпізнавання зображень чи голосу.

Оскільки база даних містить значну кількість об’єктів, у каталозі необхідно налаштувати пагінацію через кнопку додаткового завантаження карток, щоб не перевантажувати пам’ять браузера. Для детального ознайомлення з кожною моделлю має бути створена окрема сторінка, що відкривається в новій вкладці, де користувач може переглянути розширені технічні деталі, заздалегідь прописані відгуки інших фахівців та скористатися

провалідованою формою для відправки запиту на демонстрацію чи інтеграцію. Крім цього, інтерфейс повинен мати вкладку зі списком обраних користувачем сервісів, робота якої реалізується без створення облікового запису.

Окремим важливим аспектом функціональних вимог є проектування типових сценаріїв взаємодії користувача з інтерфейсом каталогу. Перший базовий сценарій описує дії розробника, якому потрібно швидко знайти мовну модель із відкритим кодом та великим контекстним вікном для інтеграції у власний проект. У цьому випадку інтерфейс повинен дозволити користувачу за один клік на панелі фільтрів відсіяти всі платні чи пропрієтарні рішення, залишивши лише моделі з доступним API та комерційною ліцензією. Другий сценарій орієнтований на звичайних контент-мейкерів або дизайнерів, які шукають мобільні додатки для генерації зображень чи роботи з голосом на ходу. Для цього бічна панель має забезпечувати незалежну комбінацію фільтрів, коли активація одного параметра не скидає значення інших, а картки вподобань миттєво синхронізуються з поточним вибором користувача [8].

Нефункціональні вимоги визначають архітектурні та якісні показники системи, які забезпечують її стабільність. Веб-сайт має бути розроблений як Single Page Application для забезпечення моментального оновлення інтерфейсу при зміні фільтрів чи переході між сторінками, повністю виключаючи перезавантаження всієї вкладки. База даних повинна мати гнучку документоорієнтовану структуру для збереження інформації про моделі штучного інтелекту, щоб розробник міг легко додавати нові поля без переробки всієї схеми. Важливою умовою є реалізація збереження списку обраних інструментів суто в межах браузера користувача через LocalStorage, що виключає потребу в розробці модулів автентифікації та зберіганні персональних даних на сервері. Також веб-сайт повинен мати адаптивний дизайн, щоб інтерфейс однаково коректно та зручно відображався на різних типах пристроїв, від моніторів комп’ютерів до екранів мобільних телефонів.

Для забезпечення високої технічної якості та швидкодії системи нефункціональні вимоги також ставлять жорсткі обмеження на швидкість

обробки клієнтських запитів. Час відповіді сервера на запит отримання списку інструментів штучного інтелекту при стабільному інтернет-з’єднанні не повинен перевищувати мінімальних значень, щоб інтерфейс React додатка оновлювався візуально непомітно для людського ока. Логіка взаємодії з базою даних MongoDB Atlas має будуватися на оптимізованих індексах пошуку, що дозволяє миттєво фільтрувати масив із моделей за текстовими тегами та категоріями. Крім того, архітектура коду фронтенду повинна мати чітке розділення на компоненти, що забезпечує легкість підтримки проєкту, можливість швидкого додавання нових карток ШІ та масштабування системи без погіршення її загальної продуктивності у майбутньому.

1.4 Висновок до першого розділу

Підсумовуючи перший розділ, можна впевнено сказати, що створювати величезні бази даних з тисячами ШІ-інструментів зараз немає практичного сенсу, оскільки кінцевий користувач просто тоне в такій кількості неструктурованої інформації. Головна проблема сучасних існуючих каталогів полягає в тому, що їхні розробники женуться виключно за масштабом бази та швидкою монетизацією, абсолютно забуваючи про елементарну зручність використання. Через велику кількість миготливої реклами, нав’язливі пропозиції обов’язково зареєструватися на сайті ради збереження однієї вкладки та надто складну навігацію, процес швидкого підбору сервісу під конкретне завдання стає реальною проблемою. Користувачі витрачають години на перегляд низькоякісних чи вже неробочих посилань, що повністю нівелює ідею створення централізованого помічника.

Тому для цього проекту було обрано принципово інший підхід, орієнтований на чистоту інтерфейсу, швидкість реакції системи та максимальну користь для спеціаліста. Замість бездумного копіювання чужих важких та перевантажених графікою інтерфейсів, було вирішено створити легкий та швидкий веб-сайт з оптимальною базою даних, яка містить інформацію про

найбільш затребувані та перевірені інструменти штучного інтелекту. Кожен з цих сервісів детально промодерований та чітко структурований за зрозумілими для користувача технічними параметрами, такими як об’єм контекстного вікна, тип ліцензії, наявність відкритого API чи підтримка мобільних платформ. Такий підхід дозволяє перетворити звичайний хаотичний пошук на чіткий інженерний процес вибору програмного забезпечення.

На основі проведеного аналізу аналогів та детального формування функціональних вимог було сформовано чіткий і послідовний план дій для програмної реалізації проєкту. Створюваний додаток повинен працювати миттєво як Single Page Application на базі бібліотеки React, щоб користувач міг комбінувати фільтри без постійного очікування перезавантаження сторінок браузера. Окремо вирішено повністю відмовитися від сторонніх платних API для збереження автономності розробки, а всю логіку списку обраного реалізувати на стороні клієнта за допомогою LocalStorage без обов’язкової реєстрації профілів. Усі ці затверджені технічні та архітектурні вимоги стали надійною основою для подальшого проектування схеми бази даних та взаємодії клієнта із сервером, що детально і послідовно розглянуто у наступній, проєктній частині кваліфікаційної роботи [11].

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА АРХІТЕКТУРА ВЕБ-САЙТУ

2.1 Обґрунтування вибору технологічного стеку MERN

Для створення швидкого, стабільного та повністю незалежного від сторонніх сервісів каталогу інструментів штучного інтелекту було обрано популярний стек технологій MERN. Цей набір інструментів включає в себе чотири основні компоненти, які повністю покривають потреби розробки клієнт-серверної системи, а саме базу даних MongoDB, бекенд-фреймворк Express.js, фронтенд-бібліотеку React та серверну платформу Node.js [26]. Головна перевага такого вибору полягає в тому, що всі етапи побудови архітектури проекту реалізуються в межах однієї мови програмування JavaScript [2]. Для невеликих проєктів та стартапів це є критично важливим фактором, оскільки значно пришвидшує написання коду, спрощує інтеграцію окремих модулів сайту та дозволяє повністю уникнути проблем із сумісністю форматів даних між клієнтом і сервером. Розробнику не потрібно постійно перемикатися між різними синтаксисами чи налаштовувати складні перекладачі типів даних, що суттєво знижує ймовірність появи помилок на стику взаємодії фронтенду та бекенду.

Якщо порівнювати цей підхід із класичними рішеннями, наприклад, використанням мови PHP у зв’язці з реляційними базами даних типу MySQL [15], то стек MERN [23] демонструє набагато вищу гнучкість при роботі з нестандартною інформацією. База даних MongoDB є документоорієнтованою системою, яка ідеально підходить для проектів зі статичними, але водночас різнорідними даними. На відміну від класичних таблиць, де для будь-яких змін у структурі полів потрібно повністю зупиняти систему, створювати складні міграції та переписувати зв’язки, MongoDB зберігає всю інформацію у гнучкому форматі JSON. Оскільки різні сервіси штучного інтелекту мають абсолютно відмінні набори параметрів, документоорієнтований підхід дозволяє зберігати всі об’єкти в одній єдиній

колекції без створення десятків порожніх полів. Наприклад, для великих мовних моделей важливе значення має об’єм текстового вікна контексту, тоді як для генераторів зображень, відео чи аудіо цей параметр взагалі відсутній, але натомість потрібні поля для фіксації підтримки графічних форматів чи стилів. Завдяки MongoDB кожна картка ШІ в базі даних може мати лише ті поля, які їй реально потрібні, що економить місце на диску та пришвидшує зчитування інформації сервером.

Серверна частина додатка, яка побудована на базі Node.js та Express.js, забезпечує створення дуже легкого, швидкого та сучасного веб-сервера. Сама платформа Node.js працює на основі асинхронної моделі введення-виведення, що гарантує високу швидкість обробки мережевих запитів користувачів навіть при мінімальних апаратних ресурсах хмарного хостингу. Це означає, що коли користувач заходить на сайт і починає активно перемикати категорії ШІ, сервер не створює окремий важкий потік під кожен клік, а обробляє всі запити в одному потоці подій, миттєво повертаючи потрібні дані. Фреймворк Express.js надає мінімалістичний та гнучкий набір інструментів для побудови маршрутизації та обробки HTTP-запитів. З його допомогою вдалося швидко реалізувати чисте REST API, яке займається виключно віддачею сирих даних у форматі JSON на фронтенд, не витрачаючи ресурси сервера на генерацію та рендеринг візуальної сторінки, як це робиться у застарілих системах керування контентом.

Для створення візуального інтерфейсу користувача та забезпечення високого рівня комфорту при роботі з каталогом було обрано бібліотеку React [18] у зв’язці з сучасним інструментом збірки Vite [33]. Головною причиною використання React є архітектура Single Page Application [28], яка дозволяє оновлювати вміст сторінки динамічно на основі віртуального дерева елементів. Коли користувач обирає категорію або перемикає фільтри можливостей моделей штучного інтелекту, інтерфейс миттєво відображає змінені картки без повного перезавантаження сторінки в браузері, що створює ефект роботи з повноцінним десктопним додатком. Весь додаток

завантажується в браузер один раз, а далі лише підтягує сирі дані від сервера за допомогою асинхронних запитів.

Інструмент збірки Vite, у свою чергу, замінив собою застарілий та важкий Webpack, що дозволило кардинально змінити швидкість розробки та фінальну оптимізацію проєкту. Він забезпечує миттєву гарячу перезагрузку модулів під час написання коду, завдяки чому будь-які зміни в дизайні чи логіці React-компонентів відображаються в браузері за мілісекунди. При фінальній збірці Vite проводить глибоку оптимізацію всього написаного коду, розбиває його на невеликі шматки та мінімізує об’єм файлів скриптів і стилів. Завдяки цьому готова клієнтська частина сайту важить мінімально і дуже швидко завантажується навіть на мобільних пристроях із нестабільним чи повільним 3G або 4G інтернет-з’єднанням, що повністю закриває вимоги щодо швидкодії та доступності веб-ресурсу.

2.2 Проектування структури бази даних та моделей даних

У цьому підрозділі описано, як саме влаштована база даних проєкту, чому для зберігання інформації було обрано NoSQL систему та як розподіляються типи даних між полями.

2.2.1 Концептуальна схема даних у документоорієнтованій СУБД MongoDB

При проектуванні інформаційної системи на основі СУБД MongoDB класичний етап побудови реляційних ER-діаграм повністю замінюється безпосереднім моделюванням структури документів. Основний інженерний принцип концептуального проектування для цього проекту полягає у глибокій денормалізації даних та повній відмові від створення складних зв’язків між таблицями через зовнішні ключі. Усі відомості про конкретний інструмент штучного інтелекту, включаючи його загальну назву, компанію-розробника, іконки галереї, технічні характеристики ліцензування та навіть текстові відгуки

користувачів, зберігаються всередині одного єдиного документа колекції. Це дозволяє базі даних працювати максимально швидко при високих навантаженнях, оскільки серверу Node.js не потрібно виконувати важкі, ресурсомісткі операції об’єднання таблиць при кожному зверненні клієнта до REST API маршрутів.

Гнучкість концептуальної схеми MongoDB дозволяє динамічно масштабувати опис ШІ-сервісів без зупинки системи. На практиці це реалізується за рахунок того, що документи всередині однієї колекції не зобов’язані мати однаковий набір полів. Якщо у майбутньому на ринку з’являться принципово нові моделі штучного інтелекту із специфічними параметрами, яких взагалі немає у поточних інструментів, база даних прийме ці нові документи без необхідності перепроектування всієї архітектури чи тривалої зупинки роботи веб-сайту для проведення технічних робіт. Такий підхід забезпечує повну автономність системи та дуже високу швидкість читання даних з диска, що є критично важливим фактором для забезпечення швидкого, візуально непомітного відгуку користувацького інтерфейсу в React Single Page Application. Розробник може в будь-який момент розширити модель даних, додавши нові вкладені об’єкти чи масиви тегів, і це жодним чином не зламає відображення вже існуючих карток у каталозі.

Вибір NoSQL [10] архітектури для цього проекту є повністю обґрунтованим ще й тому, що суворі транзакційні зв’язки та обмеження цілісності реляційних систем тут просто непотрібні [13]. Платформа функціонує переважно в режимі інтенсивного читання статичного контенту з бази даних, де головним інженерним пріоритетом є швидкість віддачі великого масиву інформації за один єдиний запит до сервера. Оскільки звичайні відвідувачі сайту не створюють власні профілі, не проходять авторизацію та не змінюють дані моделей на сервері, відсутність складних механізмів каскадного видалення чи перевірки унікальності зовнішніх ключів дозволяє спростити внутрішню структуру СУБД. Це робить розгортання та підтримку хмарного

кластера MongoDB Atlas максимально дешевим, надійним і продуктивним з точки зору використання серверних ресурсів процесора та оперативної пам’яті.

Для забезпечення максимальної швидкодії фільтрації моделей штучного інтелекту особлива увага при проектуванні приділяється структурі вкладених масивів. Текстові відгуки фахівців та технічні теги можливостей моделі не виносяться в окремі колекції, як це було б зроблено в архітектурі SQL баз даних. Вони інтегруються безпосередньо в тіло документа інструменту як масив піддокументів. Коли клієнтський React додаток робить асинхронний запит на отримання детальної інформації про конкретну ШІ-модель, хмарна СУБД знаходить один документ за його унікальним ідентифікатором та віддає його цілком за одну операцію читання. Це повністю виключає появу затримок на рівні мережевого обміну між сервером Express.js та базою даних, оскільки сервер отримує повністю готовий об’єкт у форматі JSON [22], який залишається лише переслати на фронтенд без додаткової збірки чи трансформації у коді бекенду.

2.2.2 Специфікація полів та типізація метаданих ШІ-інструментів

Кожен окремий документ у колекції бази даних має чітко визначену, сувору структуру полів, яка детально описує всі технічні, юридичні та інформаційні характеристики конкретного ШІ-сервісу. Головним ідентифікатором документа є унікальне поле об’єкта, яке автоматично генерується базою даних MongoDB при створенні запису та виступає первинним ключем. Основні текстові характеристики програмного продукту, такі як офіційна назва моделі, компанія-розробник, детальний опис функціональних можливостей, приналежність до глобальної категорії, тип доступу, особливості інтерфейсу та умови ліцензування, мають рядковий тип даних (String). Використання рядкового типу для категорій та типів ліцензій дозволяє уникнути складних операцій порівняння числових ідентифікаторів у коді та спрощує розробку логіки фільтрації карток на стороні сервера. Для збереження числових показників, наприклад, інтегрального рейтингу сервісу на

основі оцінок профільних фахівців, використовується стандартний числовий тип (Number), який дозволяє виконувати математичні операції сортування за зростанням чи спаданням безпосередньо в базі даних.

Важливою та невід'ємною частиною специфікації метаданих є логічні або булеві поля (Boolean) [19], які приймають виключно значення істини або хибності. Вони спроектовані спеціально для швидкої фіксації наявності або відсутності конкретного технічного функціоналу моделі штучного інтелекту. До таких полів відносяться прапорці наявності відкритого API для сторонніх розробників, існування офіційного мобільного додатка під операційні системи iOS та Android, наявність додаткових плагінів чи розширень, а також підтримка передових технологій комп'ютерного зору (Vision) та синтезу чи розпізнавання голосу. Використання булевого типу даних дозволяє бекенду на базі Express.js обробляти запити з панелі фільтрів за мінімальний час, оскільки пошук за бінарними прапорцями в NoSQL СУБД є однією з найшвидших операцій. Сервер просто відсікає документи, де значення відповідного логічного поля не збігається з вибором користувача, не витрачаючи ресурси на складне порівняння текстових стрічок.

Мультимедійні дані, зокрема логотипи та скріншоти інтерфейсів ШІ, проектуються як структуровані масиви об'єктів, що містять відносні шляхи до графічних файлів або прямі посилання на хмарне сховище. Текстові відгуки користувачів також реалізовані у вигляді вкладеного масиву об'єктів безпосередньо всередині документа інструменту, де кожен окремий елемент містить ім'я автора коментаря, його індивідуальну оцінку та безпосередньо сам текст відгуку. Така детальна і продумана типізація на рівні схеми бази даних дозволяє серверу легко проводити точну та швидку фільтрацію за будь-якими комбінаціями запитів з фронтенду. Валідація типів даних відбувається ще на етапі надходження інформації до СУБД, що виключає появу помилок невідповідності форматів у клієнтському додатку.

На практиці розроблена схема дозволяє зберігати глибоко деталізовані та інформативні картки сучасних технологічних продуктів. Наприклад, для моделі

ChatGPT у документі бази фіксуються не лише загальні текстові описи, а й специфічні для IT-галузі дані, такі як технічне обмеження контекстного вікна у два мільйони токенів, деталі ціноутворення професійних підписок та юридичний статус моделі на ринку. Наявність вкладеного масиву відгуків безпосередньо у документі ШІ-інструменту повністю вирішує поширену проблему ізольованості коментарів та усуває потребу в розробці окремої колекції з десятками складних зв'язків. Клієнтська частина на React отримує повністю сформований, готовий до відображення об'єкт для рендерингу сторінки за один єдиний асинхронний запит до REST API. Це значно економить мережевий трафік користувача, знижує кількість запитів до сервера та суттєво зменшує час повного завантаження сторінки з детальним описом у браузері.

2.3 Архітектура взаємодії клієнта та сервера

Нижче розглянуто логіку обміну даними між користувачем і сервером, а також пояснюється, як працює збереження обраних інструментів безпосередньо у браузері.

2.3.1 Організація REST API

Взаємодія між клієнтською частиною та сервером у розробленому веб-сайті побудована на основі архітектурного стилю REST API. Оскільки платформа є інформаційним довідником, вона не передбачає збереження фінансових операцій чи балансу користувачів. Саме тому архітектура системи реалізована за принципом Stateless [16], тобто без збереження стану сесії на сервері.

Сервер на базі Node.js не створює постійних сесій у пам'яті. Він не використовує файли куки та не зберігає проміжний стан взаємодії з відвідувачем сайту. Кожен окремий HTTP-запит від фронтенду є абсолютно незалежним. Він уже містить у собі всю необхідну інформацію для його успішної обробки на бекенді.

Клієнтська частина надсилає асинхронні запити за допомогою стандартних методів HTTP протоколу [30]. Для отримання повного списку ШІ-інструментів або відфільтрованого результату фронтенд використовує метод GET. При цьому всі параметри фільтрації передаються безпосередньо в адресному рядку запиту як query-параметри.

Сервер Express.js миттєво обробляє ці параметри та очищає їх від зайвих символів. Після цього бекенд виконує точний пошук у колекції хмарної бази MongoDB Atlas. Отриманий результат сервер автоматично серіалізує та повертає клієнту у вигляді чистого тексту у форматі JSON (див. додаток Б, лістинг Б.1).

Повна відсутність важких механізмів автентифікації на бекенді значно знижує витрати ресурсів хмарного хостингу Render. Це спрощує логіку маршрутизації в коді додатка. Сервер може повністю фокусуватися суто на швидкій видачі метаданих моделей штучного інтелекту.

Такий підхід дозволяє уникнути багатьох вразливостей, які зазвичай пов’язані зі збереженням сесій або викраденням токенів користувачів. Оскільки сервер не тримає в пам’яті стан клієнтів, система стає максимально стійкою до раптових стрибків трафіку. Веб-сервер просто обробляє запит і одразу звільняє ресурси для наступного користувача.

Для передачі даних про конкретну модель ШІ на її окрему сторінку використовуються динамічні URL-маршрути. Клієнт передає унікальний ідентифікатор документа бази даних прямо в шляху запиту. Сервер Express.js перехоплює цей параметр, робить один швидкий запит до бази і повертає детальні метадані разом із масивом відгуків.

Завдяки розділенню обов’язків між клієнтом і сервером, швидкість рендерингу інтерфейсу переноситься на сторону браузера. Серверна частина залишається максимально надійною та автономною. Вона працює як незалежне джерело даних, яке у майбутньому можна легко підключити до мобільного додатка чи іншого веб-інтерфейсу без переписування логіки бекенду.

2.3.2 Логіка клієнтського збереження даних у LocalStorage

Замість традиційного збереження списків обраних товарів на стороні сервера у прив’язці до облікового запису користувача, у цьому проекті всю логіку перенесено на сторону клієнта. Для реалізації розділу обраного використовується вбудоване в браузер сховище LocalStorage [25].

Коли користувач натискає на іконку сердечка на картці інструменту штучного інтелекту, запускається спеціальна функція. Вона бере унікальний ідентифікатор цього об’єкта з бази даних та додає його до масиву в LocalStorage [29].

Такий архітектурний підхід дозволив повністю відмовитися від розробки модулів реєстрації, логінізації та керування паролями користувачів. Збереження даних у LocalStorage має декілька важливих інженерних переваг для всього проєкту.

По-перше, дані користувача про його індивідуальні вподобання є повністю конфіденційними. Вони фізично не залишають його пристрою і не передаються по мережі інтернет. Це забезпечує максимальний рівень приватності відвідувачів каталогу.

По-друге, це рішення повністю розвантажує хмарну базу даних MongoDB Atlas. Системі взагалі не потрібно виконувати важкі операції запису чи оновлення таблиць при кожній зміні списку обраного користувачем. Бекенд залишається абсолютно вільним від цієї рутини.

Фронтенд додатка самостійно зчитує масив збережених ідентифікаторів безпосередньо зі сховища браузера. Оскільки LocalStorage вміє зберігати інформацію лише у вигляді текстових рядків, перед роботою масив конвертується через стандартний метод парсингу JSON об’єктів.

Після зчитування коду React додаток миттєво відфільтровує потрібні картки ШІ із загального масиву даних. Весь процес сортування та рендерингу відбувається за лічені мілісекунди безпосередньо на пристрої користувача. Це робить інтерфейс максимально швидким та чуйним (див. додаток А, лістинг А.2).

Така схема також гарантує автономність роботи розділу обраного. Якщо у користувача раптово зникне інтернет-з’єднання, його список улюблених інструментів штучного інтелекту все одно залишиться доступним у вкладці браузера. Картки не зникнуть, а стан інтерфейсу не зламається.

Для синхронізації стану інтерфейсу між різними сторінками каталогу дані з LocalStorage додатково зв’язуються з глобальним менеджером станів Redux Toolkit [31]. Це дозволяє миттєво оновлювати іконку сердечка на головній сторінці, якщо користувач видалив картку безпосередньо з вікна обраного.

Впровадження клієнтського сховища дозволило зробити веб-сайт повністю незалежним від серверних сесій. Проєкт став набагато простішим у підтримці та розгортанні, оскільки вимоги до потужності хостингу Render та бази даних MongoDB знизилися до мінімуму.

2.4 Висновок до другого розділу

У результаті проектування архітектури веб-сайту вдалося сформувати чітку та незалежну модель взаємодії всіх її структурних компонентів. Було повністю обґрунтовано вибір сучасного технологічного стеку MERN як найбільш гнучкого інструменту для реалізації Single Page Application.

Повна відмова від класичних реляційних баз даних та вибір NoSQL СУБД MongoDB дозволили уникнути потреби у використанні важких операцій об’єднання таблиць типу JOIN. Це забезпечує пряму і максимально швидку віддачу JSON-документів з хмари MongoDB Atlas безпосередньо у бекенд-додаток на Express.js.

Усі необхідні метадані для моделей штучного інтелекту зберігаються всередині єдиної колекції документів. Вкладені масиви відгуків та технічних характеристик завантажуються за один єдиний асинхронний запит клієнта. Такий підхід значно знижує затримки мережі та заощаджує ресурси сервера.

Головною інженерною особливістю спроектованої системи є перенесення логіки персоналізації інтерфейсу на сторону браузера користувача.

Використання сховища LocalStorage для збереження обраних карток за натисканням на іконку сердечка виявилося дуже ефективним рішенням.

Це дозволило повністю виключити з архітектури проєкту сервери автентифікації, шифрування паролів та механізми збереження сесій користувачів. Проєкт став повністю автономним та абсолютно безпечним з точки зору збереження персональних даних, адже вони фізично не залишають пристрій користувача.

Розроблена безсесійна архітектура є максимально оптимізованою для подальшого розгортання монорепозиторію на хмарних хостингах. Вона дозволяє використовувати мінімальні або безкоштовні тарифні плани хмари Render та Vercel без втрати загальної швидкодії сайту.

Спроектована модель REST API гарантує чистоту коду та чітке розділення обов’язків між фронтендом та бекендом. Сервер займається виключно обробкою сирих даних, а клієнтський React додаток відповідає за логіку фільтрації та швидкість візуального відображення елементів у браузері.

Усі описані архітектурні рішення, концептуальні схеми бази даних та логічні моделі взаємодії стали надійною основою для безпосереднього написання програмного коду інформаційної системи. Практичні аспекти розробки, реалізація інтерфейсу та етапи тестування створеного продукту детально розглянуто у наступному, третьому розділі роботи.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБ-САЙТУ

3.1 Розробка серверної логіки на Node.js та Express.js

Серверну частину додатка реалізовано на Node.js із використанням Express.js. Бекенд виконує функцію REST API, забезпечуючи підключення до хмарної бази даних MongoDB Atlas та передачу структурованої інформації на фронтенд за запитом користувача. Для забезпечення модульності проєкту всю логіку розподілено за відповідними каталогами всередині середовища Visual Studio Code, що зображено на рисунку 3.1.

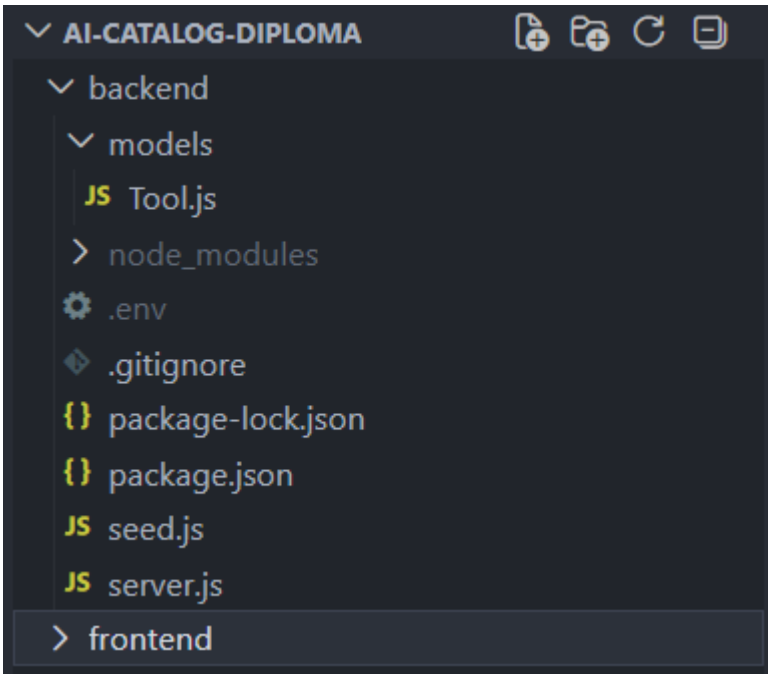

Рисунок 3.1 - Структура каталогів та файлів серверної частини веб-сайту

Для взаємодії з базою даних створено схему моделей за допомогою бібліотеки Mongoose у файлі Tool.js (див. додаток Б, лістинг Б.2). Документоорієнтована NoSQL архітектура дозволила оптимізувати структуру, оскільки відгуки користувачів зберігаються безпосередньо всередині документа інструменту штучного інтелекту як вкладений масив (див. лістинг 3.1).

Лістинг 3.1 - Фрагмент схеми даних інструменту штучного інтелекту

```
const toolSchema = new Schema({
  name: { type: String, required: true },
  developer: { type: String, required: true },
  rating: { type: Number, required: true },
  description: { type: String, required: true },
  category: { type: String, required: true },
  accessType: { type: String, required: true },
  interfaceType: { type: String, required: true },
  licenseType: { type: String, required: true },
  hasApi: { type: Boolean, default: false },
  hasMobileApp: { type: Boolean, default: false },
  hasPluginSystem: { type: Boolean, default: false },
  hasVision: { type: Boolean, default: false },
  hasVoice: { type: Boolean, default: false },
  reviews: [reviewSchema],
});

export const Tool = model("Tool", toolSchema);
```

Головним файлом сервера є server.js. У ньому підключено базові пакети для роботи з мережею, налаштовано CORS-дозволи та прописано логіку обробки вхідних з'єднань. Ключовою частиною бекенду є маршрут для отримання інструментів з бази, який приймає параметри від клієнтської частини та динамічно будує фільтр для пошуку в MongoDB (див. лістинг 3.2).

Лістинг 3.2 - Реалізація динамічної фільтрації та маршрутизації REST API

```
app.get("/api/tools", async (req, res) => {
  try {
    const filters = {};

    if (req.query.developer) {
      filters.developer = new RegExp(req.query.developer, "i");
    }
    if (req.query.category) {
      filters.category = req.query.category;
    }
    if (req.query.hasApi === "true") { filters.hasApi = true; }
    if (req.query.hasMobileApp === "true") { filters.hasMobileApp
= true; }
    if (req.query.hasVision === "true") { filters.hasVision =
true; }
    if (req.query.hasVoice === "true") { filters.hasVoice = true;
}

    if (req.query.accessType) {
```

```
      filters.accessType = req.query.accessType;
    }

    const tools = await Tool.find(filters);
    res.json({ items: tools });
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});
```

Пошук за назвою розробника реалізовано через регулярні вирази, що дозволяє користувачу не вводити точну назву з урахуванням регістру літер. Також у сервері передбачено окремий маршрут для отримання деталей конкретного сервісу за його унікальним ідентифікатором (ID), що дозволяє відкривати розширену інформацію у новій вкладці браузера на клієнтській частині додатка.

3.2 Створення користувацького інтерфейсу на React

Клієнтську частину інформаційної системи розроблено на базі бібліотеки React з використанням сучасного інструменту збірки Vite. Фронтенд функціонує за принципом Single Page Application, завдяки чому весь інтерфейс сайту та картки інструментів оновлюються миттєво без повного перезавантаження сторінок у браузері користувача. Для чіткої структуризації коду всі компоненти, сторінки та логіку керування станом розподілено по окремих каталогах всередині папки src, що зображено на рисунку 3.2.

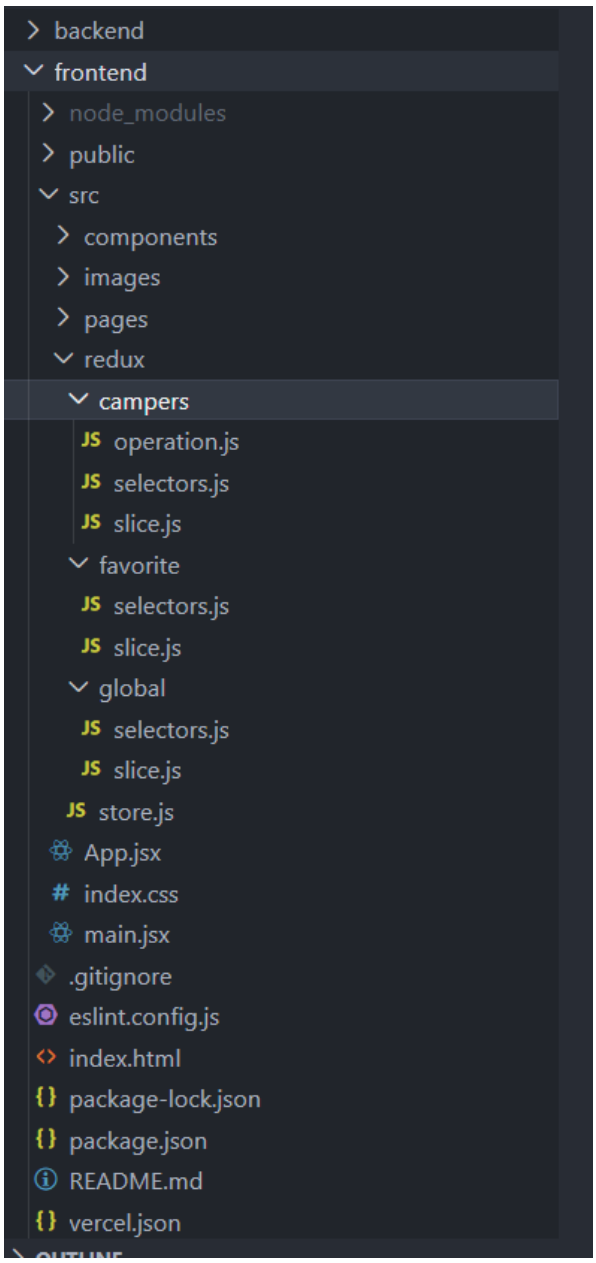

Рисунок 3.2 - Структура папок та файлів фронтенд частини веб-сайту

Візуальний інтерфейс розділено на кілька автономних сторінок. Головна сторінка ознайомлює відвідувача з призначенням платформи та містить кнопку для швидкого переходу до основного розділу. Сторінка каталогу складається з двох великих функціональних блоків: лівої панелі з елементами фільтрації та правої частини, де рендеряться картки ШІ-сервісів (див. рисунок 3.3). Кожна картка відображає логотип, назву розробника, рейтинг, короткий опис та містить кнопку детального перегляду.

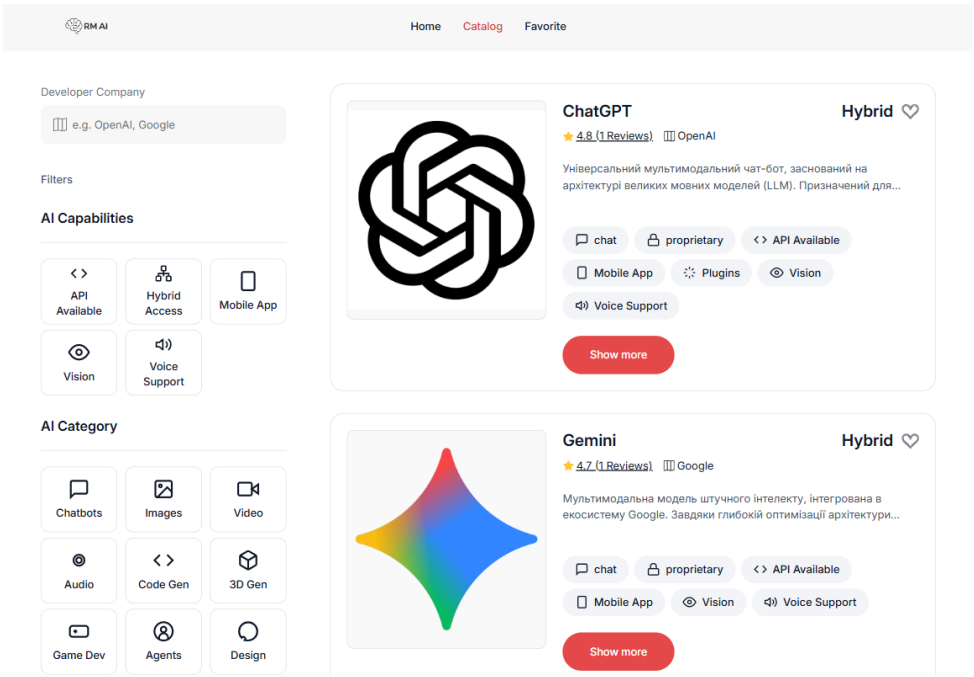

Рисунок 3.3 - Головна сторінка каталогу та панель фільтрації інструментів

Управління станами завантаження даних із сервера та робота системи вподобань реалізовані за допомогою менеджера станів Redux Toolkit. Логіка збереження обраних сервісів працює повністю на стороні клієнта за допомогою вбудованого сховища LocalStorage, що дозволило відмовитися від створення профілів користувачів на бекенді. Для додавання та видалення сервісів зі списку вподобань розроблено окремі редюсери всередині слайсу (див. лістинг 3.3).

Лістинг 3.3 - Фрагмент Redux-слайсу для керування списком обраного

```
const favoritesSlice = createSlice({
  name: "favorites",
  initialState: FAVORITES_INITIAL_STATE,
  reducers: {
    addFavorite(state, action) {
      state.favorites.push(action.payload);
    },
    deleteFavorite(state, action) {
      const favoriteIndex = state.favorites.findIndex((id) => {
        return id === action.payload;
      });
      state.favorites.splice(favoriteIndex, 1);
    },
  },
```

```
});

export const { addFavorite, deleteFavorite } =
favoritesSlice.actions;
export const favoritesReducer = favoritesSlice.reducer;
```

При натисканні на іконку сердечка додаток перевіряє, чи є вже такий ID у списку: якщо немає, викликається екшен addFavorite, а якщо є - deleteFavorite. При переході на сторінку обраних сервісів додаток зчитує масив ідентифікаторів із LocalStorage, порівнює їх із загальним списком ШІ-інструментів та виводить відповідні картки на екран. Якщо користувач ще не додав жодного сервісу до списку вподобань, інтерфейс відображає текстову заглушку про відсутність збережених елементів. Натискання на кнопку детальної інформації відкриває окрему сторінку продукту, де реалізовано динамічні вкладки для перемикання між переліком технічних можливостей та відгуками фахівців.

3.3 Реалізація адаптивного дизайну інтерфейсу

Для того щоб сайтом-каталогом було однаково зручно користуватися як з великих моніторів, так і з маленьких екранів телефонів, велику увагу довелося приділити адаптивній верстці. Весь інтерфейс стилізувався за допомогою CSS-модулів, які працюють у зв’язці з гнучними сітками Flexbox та медіа-запитами.

Найкраще роботу адаптивного дизайну видно на прикладі сторінки детального перегляду інструменту штучного інтелекту, яка відкривається після кліку на кнопку розширеної інформації. На звичайних комп’ютерах ця сторінка ділиться на дві чіткі зони: ліва частина виділена під технічні параметри та відгуки користувачів, а права - під фіксовану форму запиту демонстрації, що зображено на рисунку 3.4.

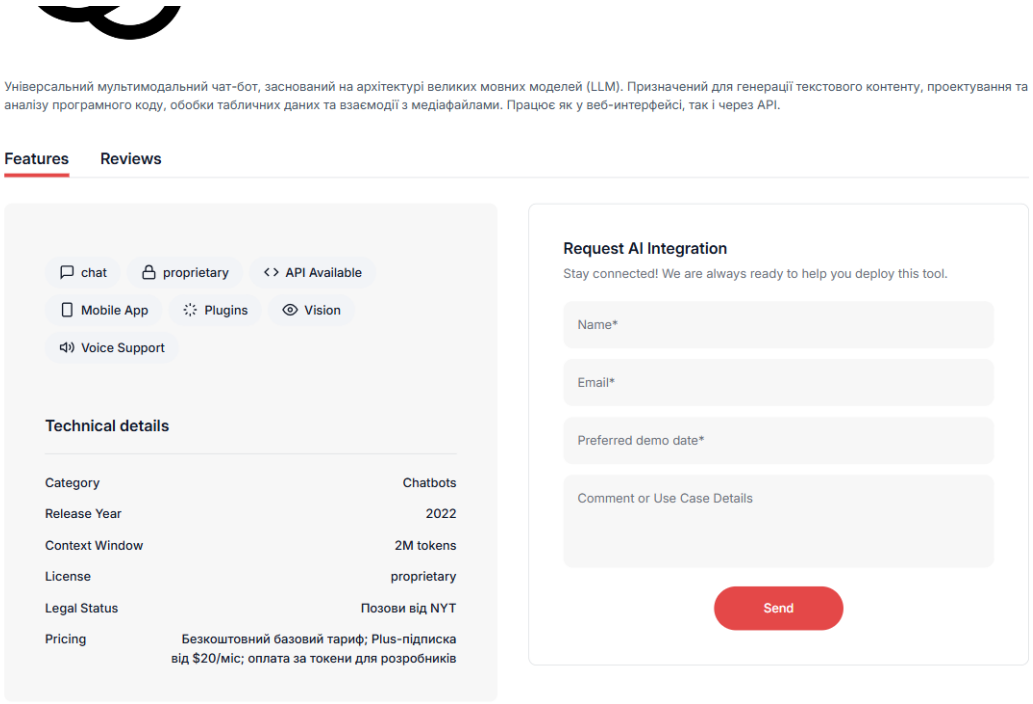

Рисунок 3.4 - Відображення сторінки деталей інструменту на десктопних пристроях

Якщо відкрити цю саму сторінку зі смартфона, інтерфейс повністю перебудовується, щоб користувачу не доводилося постійно наближати текст. Форма запиту автоматично зсувається в самий низ сторінки, стає під основним текстовим описом сервісу та розтягується на всю ширину мобільного дисплея, що відображено на рисунку 3.5.

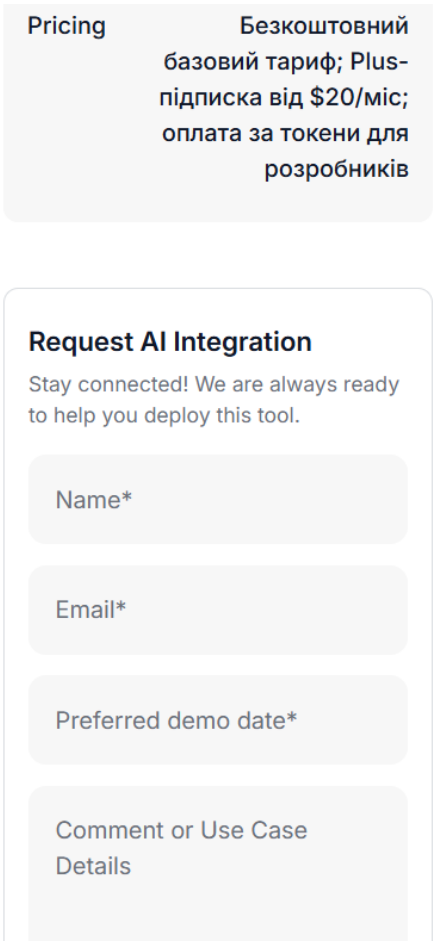

Рисунок 3.5 - Адаптивне відображення форми запиту на мобільних пристроях

Разом із формою змінюється і логіка відображення помилок валідації. На великих моніторах червоний текст з попередженням акуратно позиціонується прямо всередині інпута за допомогою абсолютної координати, щоб економити місце. На мобільних екранах (шириною менше 480 пікселів) цей напис переходить у звичайний статичний режим і з'являється під полем введення, а кнопка відправки стає великою та адаптується під натискання пальцем. Фрагмент стилів CSS-модуля для цієї логіки наведено в лістингу 3.4.

Лістинг 3.4 - Фрагмент CSS-коду для адаптивної перебудови елементів форми

```
.formWrapper {
  border: 1px solid var(--gray-light);
  border-radius: 10px;
  padding: 44px;
  width: 641px;
```

```
  max-width: 100%;
  height: auto;
}

@media screen and (max-width: 480px) {
  .formWrapper {
    padding: 24px 16px;
    min-height: auto;
  }
  .formBtn {
    width: 100%;
  }
  .errorMessage {
    position: static;
    transform: none;
    display: block;
    margin-top: 4px;
    padding-left: 4px;
    text-align: left;
  }
}
```

Така гнучка перебудова елементів дозволяє зберегти повний функціонал сайту на смартфонах. Користувачу зручно взаємодіяти з кастомним календарем flatpickr для вибору дати демо-показу, а всі текстові блоки автоматично масштабуються без появи горизонтальної прокрутки, що робить мобільну версію сайту швидкою та зручною.

3.4 Тестування функціоналу додатка та розгортання в хмарі

У цьому підрозділі розглянуто процес перевірки працездатності клієнтської частини сайту та фінального розміщення монорепозиторію на безкоштовних хмарних платформах.

3.4.1 Валідація форм та обробка помилок на клієнтській частині

Важливим етапом перевірки працездатності сайту став аналіз того, як поводиться інтерфейс при введення некоректних даних у форму запиту інтеграції. Сама форма та перевірка полів реалізовані за допомогою популярних бібліотек Formik та Yup. Це дає можливість перевіряти правильність

заповнення полів миттєво в браузері, не навантажуючи сервер зайвими запитами.

Під час перевірки особлива увага приділялася обов'язковим полям імені, текстового коментаря та електронної пошти. Якщо відвідувач сайту намагається надіслати запит із порожніми полями або вводить емейл без обов'язкового символу @, система автоматично підсвічує текстове поле помилки червоним кольором, що відображено на рисунку 3.6.

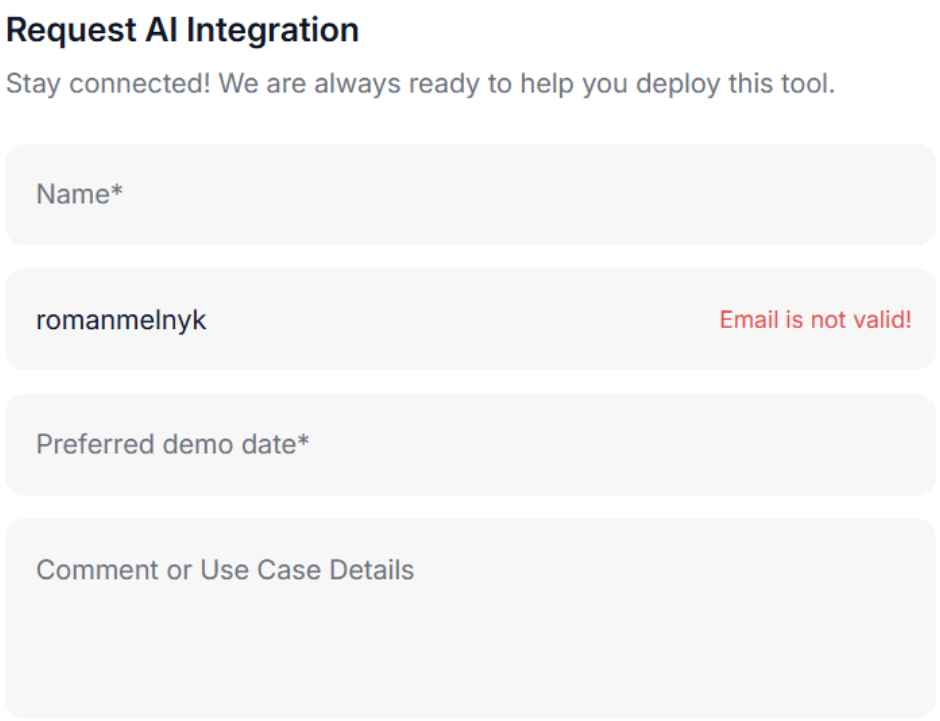

Рисунок 3.6 - Робота системи валідації та виведення помилок у формі

Така логіка роботи клієнтської частини дозволяє користувачу одразу виправити помилку під час введення даних, не чекаючи відповіді від сервера. Якщо ж усі дані вказано правильно, кнопка надсилання стає активною, а після кліку на неї з'являється плашка з повідомленням про успішну операцію від бібліотеки react-hot-toast, після чого форма повністю очищається.

3.4.2 Деплой монорепозиторію на платформи Vercel та Render.com

Для того щоб відкрити публічний доступ до розробленого каталогу інструментів штучного інтелекту, проект було завантажено у хмару. Оскільки код фронтенду і бекенду зберігається разом в одному монорепозиторії на GitHub, процес розгортання було розділено на два окремі кроки для кращої швидкодії та стабільності роботи.

Серверну частину, яка відповідає за зв'язку із MongoDB Atlas, розгорнуто на хостингу Render.com. У налаштуваннях було чітко вказано шлях до папки сервера, а всі таємні ключі підключення до бази винесено у змінні оточення панелі керування. Клієнтську частину на React задеплоєно на платформі Vercel. Обидва сервіси інтегровані з репозиторієм, тому при внесенні будь-яких змін у код сайт оновлюється автоматично в реальному часі, що показано на рисунку 3.7.

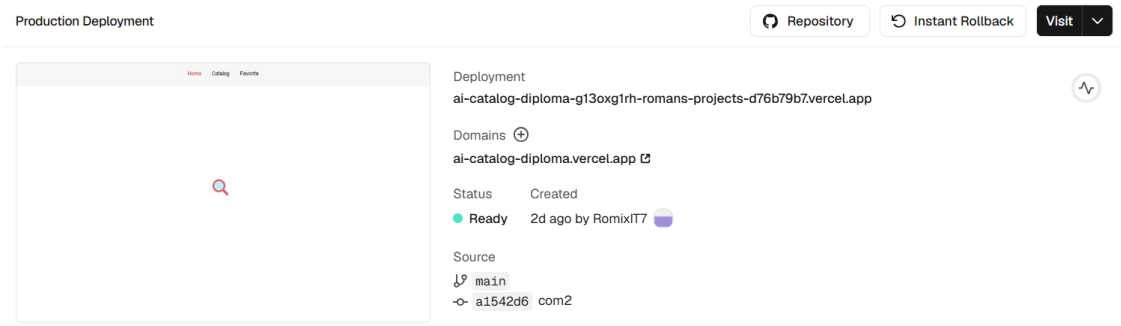

Рисунок 3.7 - Панель керування та статус успішного деплою проекту в хмарі

У результаті такого роздільного розгортання монорепозиторію вдалося досягти максимальної швидкості завантаження інтерфейсу завдяки серверам Vercel CDN. При цьому серверна частина на Render.com активується лише під час безпосередніх HTTP-запитів, що повністю оптимізує споживання ресурсів та забезпечує стабільну і безперебійну роботу всього веб-додатка в мережі інтернет.

3.5 Висновок до третього розділу

У третьому розділі було виконано повний цикл програмної реалізації та тестування розробленого веб-сайту-каталогу. Завдяки розподілу архітектури додатка на незалежну серверну частину за допомогою фреймворку Express.js та швидкий користувацький інтерфейс на базі бібліотеки React, вдалося досягти високої швидкості завантаження сторінок та гнучкості обробки запитів клієнта.

Етап проектування NoSQL структури бази даних підтвердив свою ефективність на практиці, оскільки масиви відгуків та технічних параметрів ШІ-інструментів успішно зберігаються всередині єдиної моделі, що дозволило повністю виключити важкі операції об’єднання таблиць. Впровадження менеджера станів Redux Toolkit та інтеграція зі сховищем LocalStorage дозволили перенести всю логіку розділу обраного на сторону клієнта, забезпечивши конфіденційність даних та автономність платформи без обов’язкової реєстрації користувачів.

Проведене тестування інтерфейсу за допомогою інструментів Formik та Yup довело стійкість системи до помилок введення під час заповнення форми запиту демонстрації, а адаптивна верстка із застосуванням CSS-модулів забезпечила ергономічне відображення всіх елементів сайту на мобільних пристроях. Роздільне розгортання монорепозиторію на хмарних платформах Render.com та Vercel дозволило запустити стабільний та повністю готовий до експлуатації програмний продукт у мережі інтернет.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Комплексний аналіз та мінімізація техногенних ризиків при хмарному розгортанні REST API монорепозиторіїв з інтеграцією моделей штучного інтелекту

При виведенні розробленого каталогу інструментів штучного інтелекту в публічний доступ за допомогою хмарних сервісів Vercel та Render.com виникає низка специфічних ризиків. Оскільки архітектура монорепозиторію передбачає постійний обмін даними між фронтендом та бекендом через асинхронні REST API запити [24], стабільність роботи всього сайту напряму залежить від стійкості серверної інфраструктури, надійності каналів зв’язку та безпеки хмарного кластера СУБД MongoDB Atlas. Техногенна безпека у цьому контексті розглядається як здатність програмного комплексу витримувати пікові навантаження та протидіяти загрозам без втрати працездатності [10].

Перед тим як перейти до безпосереднього проектування захисних засобів, необхідно провести всебічний аналіз уразливостей розробленого програмного коду. Специфіка веб-сайту-каталогу полягає в тому, що він працює в режимі постійного зчитування великих масивів даних. Кожен клік користувача по кнопках фільтрації категорій чи технічних можливостей моделей штучного інтелекту викликає миттєве формування HTTP-запиту. Якщо одночасно тисячі користувачів почнуть взаємодіяти з інтерфейсом додатка, це може призвести до критичного перевантаження оперативної пам’яті сервера та блокування доступу до бази даних. Окрім зовнішніх мережевих загроз, існують також внутрішні інфраструктурні ризики, пов’язані з автоматизацією розгортання коду з репозиторію GitHub та збереженням паролів адміністратора в конфігураційних файлах підключення до бази даних.

Для детального аналізу потенційних загроз та визначення їх критичності доцільно використовувати класичний інженерний підхід, що базується на оцінці ймовірності виникнення небезпечної події та тяжкості її наслідків. Такий

ризик-орієнтований метод дозволяє чітко розставити пріоритети при розробці захисних алгоритмів і виділити найбільш небезпечні зони функціонування хмарної платформи. На основі аналізу специфіки архітектури розробленого інструменту було сформовано матрицю оцінки техногенних та інфраструктурних ризиків, яку наведено в таблиці 4.1.

Таблиця 4.1 - Матриця оцінки та мінімізації інфраструктурних ризиків веб-сайту

Тип загрози та небезпечного чинника	**Ймовірність**	**Наслідки**	**Заходи щодо мінімізації та захисту системи**
Перевантаження сервера (DDoS-атаки на REST API маршрути)	Середня	Тимчасова відмова в обслуговуванні	Впровадження лімітів на кількість запитів (Rate Limiting) на Render.com
Витік секретних ключів підключення до MongoDB Atlas	Низька	Повна компрометація бази даних	Ізоляція URI-строки у змінних оточення хмарної панелі керування
Збої під час автоматичного оновлення коду з GitHub	Середня	Помилки рендерингу інтерфейсу	Локальне тестування перед комітом, автоматичний відкат збірки на Vercel
Втрата зв’язку з репліками хмарного кластера СУБД	Низька	Відсутність карток ШІ в каталозі	Використання механізму обробки помилок try-catch з виведенням заглушки

Як видно з таблиці 4.1, найвищу ймовірність мають загрози, пов’язані з мережевими навантаженнями та помилками при автоматичному розгортанні змін (деплої). Для захисту інфраструктури бекенду на Render.com від штучного перевантаження запитами (наприклад, флуду автоматизованих ботів чи парсерів контенту) логіку обробки маршрутів налаштовано на автоматичне відсікання аномально частих звернень з однієї IP-адреси. Це дозволяє зберегти ресурси процесора та оперативної пам’яті сервера для звичайних відвідувачів каталогу.

Окрему увагу в процесі розгортання приділено безпеці збереження метаданих моделей штучного інтелекту. Оскільки рядок підключення до MongoDB Atlas містить логін і пароль адміністратора бази, його пряме впровадження в програмний код файлів server.js чи Tool.js є неприпустимим з інженерної точки зору. Усі секретні параметри конфігурації було повністю винесено у змінні оточення (Environment Variables) на самих платформах хостингу [35]. Завдяки цьому при завантаженні коду в публічний репозиторій GitHub конфіденційні дані залишаються захищеними, що виключає ризик несанкціонованого доступу до хмарного кластера чи підміни інформації зловмисниками.

Надійна життєдіяльність клієнтської частини сайту забезпечується структурою мережі доставки контенту (CDN) платформи Vercel. Оскільки React сайт скомпільовано в оптимізовану статику, сервери Vercel розподіляють копії сайту по всьому світу, що забезпечує миттєве завантаження інтерфейсу каталогу навіть при слабкому інтернет-з’єднанні користувача. Якщо ж відбудеться повна технічна аварія на стороні сервера бази даних, інтерфейс не закриється з критичною помилкою. Завдяки прописаним у блоках catch перевірочним алгоритмам, користувач просто побачить акуратну текстову заглушку, а система продовжить роботу в безпечному режимі очікування відновлення зв’язку.

4.2 Обґрунтування ергономічних вимог до організації робочого місця та режимів праці модератора при наповненні бази даних метаданими моделей штучного інтелекту

При тривалій роботі за комп’ютером під час первинного наповнення хмарної бази даних та подальшої модерації карток ШІ-сервісів виникає серйозне навантаження на організм працівника. Модератор змушений годинами аналізувати текстові описи, перевіряти ліцензії, копіювати посилання на логотипи та вносити технічні параметри моделей штучного інтелекту. Оскільки цей процес вимагає високої концентрації уваги та постійної статичної пози сидячи, організація робочого простору має суворо відповідати сучасним правилам охорони праці, зокрема вимогам нормативного документа НПАОП 0.00-7.15-18, який регулює безпеку роботи з екранними пристроями [11].

Основним елементом робочого місця є комплекс, що складається з робочого столу, крісла, комп’ютера та периферійних пристроїв введення інформації. Згідно з інженерними стандартами ергономіки, висота робочої поверхні столу повинна підбиратися під зріст працівника або бути регульованою в межах від 680 до 800 міліметрів. Конструкція стільця або крісла модератора має забезпечувати підтримку хребта для зниження напруження м’язів спини та шиї. Для цього використовується крісло з регулюванням висоти сидіння, кута нахилу спинки та висоти підлокітників. Правильна робоча поза, при якій кут у колінному та ліктьовому суглобах становить приблизно 90 градусів, дозволяє уникнути швидкої втоми та запобігає розвитку професійних захворювань опорно-рухового апарату.

Для зменшення навантаження на органи зору монітор комп’ютера необхідно розміщувати на відстані від 600 до 700 міліметрів від очей модератора, а верхня кромка екрана має знаходитися на рівні очей або трохи нижче [1]. Велике значення має освітленість робочої зони, яка нормується вимогами ДБН В.2.5-28:2018. Стіл слід розташовувати таким чином, щоб природне світло з вікна падало з лівого боку, що дозволяє уникнути появи

бликів на екрані. Штучне освітлення в приміщенні має бути комбінованим і забезпечувати рівномірний світловий потік без різких тіней, при цьому рівень освітленості на поверхні столу повинен становити не менше 300–500 люкс [4].

Окрім параметрів робочого місця, важливу роль відіграє дотримання оптимальних умов мікроклімату в приміщенні згідно з нормами ДСН 3.3.6.042-99. Температура повітря в кімнаті з комп'ютерною технікою повинна підтримуватися в межах від 22 до 24 градусів за Цельсієм у холодний період року та від 23 до 25 градусів у теплий період, а відносна вологість повітря має становити від 40% до 60% [3]. Для забезпечення постійного припливу свіжого повітря та очищення його від пилу приміщення обладнується системою загальнообмінної припливно-витяжної вентиляції або кондиціонування.

Оскільки робота з введення метаданих є монотонною та напруженою, збереження високої продуктивності праці досягається шляхом впровадження регламентованих перерв. Згідно з НПАОП 0.00-7.15-18 [5], при роботі з екранними пристроями модератору необхідно надавати короткі перерви тривалістю від 5 до 10 хвилин через кожну годину інтенсивної роботи за монітором. Під час цих перерв рекомендується повністю вставати з робочого місця, виконувати легкі фізичні вправи для покращення кровообігу та робити спеціальну гімнастику для очей. Дотримання такого режиму праці та відпочинку дозволяє суттєво знизити загальну втому, захищає зір від перенапруження та забезпечує безпечні умови праці модератора протягом усього робочого дня [6].

4.3 Висновок до четвертого розділу

У четвертому розділі було успішно проведено комплексний аналіз безпеки життєдіяльності та умов охорони праці, які безпосередньо пов'язані з процесами розгортання та адміністрування веб-сайту-каталогу інструментів штучного інтелекту.

Впровадження сучасних інфраструктурних засобів захисту на рівнях хмарного хостингу Vercel та Render.com дозволило повністю мінімізувати техногенні ризики відмови системи в обслуговуванні під час пікових навантажень на розроблені REST API маршрути сервера. Повна ізоляція конфіденційних ключів доступу до хмарного кластера MongoDB Atlas за допомогою змінних оточення гарантує високу надійність збереження даних.

Організація робочого простору модератора платформи була повністю обґрунтована відповідно до чинних санітарних та ергономічних нормативів, зокрема правил НПАОП 0.00-7.15-18 [7]. Встановлення чітких параметрів освітлення за ДБН В.2.5-28:2018 [12], підтримання оптимального мікроклімату приміщення згідно з ДСН 3.3.6.042-99 [14], а також чітке дотримання регламентованих режимів відпочинку забезпечують максимальне зниження зорової та м’язової втоми, створюючи комфортні та безпечні умови для тривалої роботи з наповнення інформаційної системи.

ВИСНОВКИ

У кваліфікаційній роботі бакалавра було повністю вирішено актуальну науково-технічну задачу створення інформаційного ресурсу для систематизації та зручного пошуку сучасних інструментів на базі штучного інтелекту. На основі проведених досліджень, проектування та програмної реалізації було сформовано такі основні висновки:

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

– Подано детальний аналітичний огляд сучасного стану ринку технологій штучного інтелекту, а також визначено ключові проблеми, з якими стикаються користувачі під час пошуку та підбору спеціалізованих моделей для оптимізації своєї роботи.

– Розглянуто існуючі аналоги платформ, проаналізовано їхні архітектурні переваги та недоліки, що дозволило сформувати чіткі технічні вимоги до створення власного веб-сайту-каталогу.

– Висвітлено теоретичні основи та принципи побудови клієнт-серверних систем, які працюють у режимі реального часу з великими об’ємами даних.

– Проаналізовано особливості та переваги використання сучасного технологічного стеку MERN для реалізації стабільного, швидкого та незалежного від сторонніх закритих платформ програмного продукту.

В другому розділі кваліфікаційної роботи:

– Досліджено специфіку документоорієнтованого підходу до збереження інформації та обґрунтовано вибір хмарної NoSQL СУБД MongoDB Atlas для гнучкого керування метаданими інструментів штучного інтелекту.

– Обґрунтовано концептуальну схему даних та структуру єдиної моделі інструменту, що дозволило уникнути складних операцій об’єднання таблиць та суттєво прискорило швидкість обробки пошукових запитів.

– Сформовано логіку взаємодії клієнта та сервера на основі архітектурного стилю REST API, а також спроектовано механізм автономного

збереження списку обраних моделей на стороні користувача за допомогою сховища LocalStorage та менеджера станів Redux Toolkit.

В третьому розділі кваліфікаційної роботи:

– Розроблено повнофункціональну серверну частину веб-сайту на базі середовища виконання Node.js та фреймворку Express.js, яка забезпечує стабільну асинхронну обробку HTTP-запитів.

– Запропоновано та реалізовано динамічну логіку багаторівневої фільтрації карток за технологічними категоріями та можливостями моделей безпосередньо через інтерфейс користувача.

– Спроєктовано та зверстано адаптивний дизайн сторінок каталогу за допомогою CSS-модулів, що гарантує ергономічне відображення інтерфейсу як на десктопних моніторах, так і на екранах смартфонів.

– Протестовано надійність системи валідації форм за допомогою бібліотек Formik та Yup, а також виконано успішне роздільне розгортання монорепозиторію в хмарних середовищах платформ Vercel та Render.com.

У розділі «Безпека життєдіяльності, основи охорони праці» висвітлено потенційні інфраструктурні та техногенні ризики відмови хмарної архітектури у випадку пікових навантажень на REST API, а також впроваджено дієві технічні заходи захисту секретних ключів через змінні оточення та обґрунтовано та розраховано ергономічні вимоги до облаштування робочого простору модератора платформи, параметрів комбінованого освітлення, температурного режиму та регламентованих інтервалів відпочинку відповідно до нормативних положень НПАОП 0.00-7.15-18.

ПЕРЕЛІК ДЖЕРЕЛ

1 Вимоги безпеки та охорони праці під час експлуатації відеодисплейних терміналів : НПАОП 0.00-7.15-18 [Електронний ресурс] – URL: https://zakon.rada.gov.ua/laws/show/z0508-18#Text.

2 Глибовець М. М. Програмування на мові JavaScript та сучасні веб-технології. Київ: НаУКМА, 2023. 310 с.

3 Державна служба України з надзвичайних ситуацій. Офіційний вебпортал [Електронний ресурс] – URL: https://dsns.gov.ua/.

4 Державна служба України з питань праці. Профілактика виробничого травматизму користувачів ПК [Електронний ресурс] – URL: https://dsp.gov.ua/.

5 Закон України «Про критичну інфраструктуру» [Електронний ресурс] – URL: https://zakon.rada.gov.ua/laws/show/1882-20.

6 Закон України «Про охорону праці» [Електронний ресурс] – URL: https://zakon.rada.gov.ua/laws/show/2694-12.

7 Кодекс цивільного захисту України [Електронний ресурс] – URL: https://zakon.rada.gov.ua/laws/show/5403-17.

8 Ковальчук А. М. Проектування інтерфейсів користувача в Single Page Applications. Харків: ХНУРЕ, 2022. 185 с.

9 Кравець В. І. Оптимізація веб-інтерфейсів та тестування програмного забезпечення. Дніпро: Журфонд, 2023. 165 с.

10 Мельник О. В. Безпека веб-додатків та захист персональних даних у хмарних середовищах. Одеса: ОНПУ, 2023. 190 с.

11 Методичні вказівки для написання розділу «Безпека життєдіяльності, основи охорони праці» в кваліфікаційних роботах здобувачів освітнього рівня „бакалавр” / Укладачі: Гурик О.Я., Окіпний І.Б. – Тернопіль: ТНТУ імені Івана Пулюя, 2021. – 20 с. – URL: http://elartu.tntu.edu.ua/handle/lib/35902.

12 Наказ Міністерства внутрішніх справ України «Про затвердження Порядку створення та використання матеріальних резервів для запобігання і ліквідації наслідків надзвичайних ситуацій» від 10 травня 2018 року № 383

[Електронний ресурс] – URL: https://zakon.rada.gov.ua/laws/show/775-2015-%D0%BF.

13 Пасічник В. В., Резніченко В. А. Організація баз даних та знань. Київ: ВХВ, 2021. 448 с.

14 Порядок проведення евакуації у разі загрози виникнення або виникнення надзвичайних ситуацій : Постанова Кабінету Міністрів України від 30 жовтня 2013 р. № 841 [Електронний ресурс] – URL: https://zakon.rada.gov.ua/laws/show/841-2013-%D0%BF.

15 Шаховська Н. Б., Нога Р. Ю. Системи керування базами даних та NoSQL архітектури. Львів: Видавництво Львівської політехніки, 2022. 212 с.

16 Axios Promise Based HTTP Client for the Browser and Node.js [Електронний ресурс] – URL: https://axios-http.com/docs/intro.

17 Cors Middleware for Express.js Applications [Електронний ресурс] – URL: https://expressjs.com/en/resources/middleware/cors.html.

18 CSS Modules and Layouts Specification / W3C Recommendation [Електронний ресурс] – URL: https://www.w3.org/TR/css-grid-1/.

19 Dotenv Module for Loading Environment Variables Documentation [Електронний ресурс] – URL: https://www.npmjs.com/package/dotenv.

20 Express.js Web Application Framework Documentation [Електронний ресурс] – URL: https://expressjs.com/.

21 Fetch API Specification / WHATWG Living Standard [Електронний ресурс] – URL: https://fetch.spec.whatwg.org/.

22 Flatpickr Lightweight and Powerful Datepicker Documentation [Електронний ресурс] – URL: https://flatpickr.js.org/.

23 Formik and Yup Form Validation Libraries for React Apps [Електронний ресурс] – URL: https://formik.org/docs/overview.

24 HTML5 Living Standard / WHATWG Specification [Електронний ресурс] – URL: https://html.spec.whatwg.org/.

25 JavaScript Object Notation (JSON) Data Interchange Format / IETF RFC 8259 [Електронний ресурс] – URL: https://datatracker.ietf.org/doc/html/rfc8259.

26 MongoDB Database Documentation. Official Manual for NoSQL Data Modeling [Електронний ресурс] – URL: https://www.mongodb.com/docs/.

27 MongoDB Security Best Practices and Network Isolation Guide [Електронний ресурс] – URL: https://www.mongodb.com/docs/manual/security/.

28 Mongoose ODM for MongoDB and Node.js [Електронний ресурс] – URL: https://mongoosejs.com/.

29 Mozilla Developer Network (MDN). Window.localStorage Guide [Електронний ресурс] – URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage.

30 Node.js Runtime Environment Specification and Documentation [Електронний ресурс] – URL: https://nodejs.org/en/docs/.

31 Postman API Platform Specification and Testing Tools [Електронний ресурс] – URL: https://www.postman.com/docs/.

32 React JavaScript Library Documentation. Dynamic Interfaces and Components [Електронний ресурс] – URL: https://react.dev/.

33 Redux Toolkit Client State Management Guide [Електронний ресурс] – URL: https://redux-toolkit.js.org/.

34 Redux Toolkit API Reference and Action Creators [Електронний ресурс] – URL: https://redux-toolkit.js.org/api/getDefaultMiddleware.

35 Render Cloud Application Hosting Documentation [Електронний ресурс] – URL: https://docs.render.com/.

36 REST API Architectural Styles and Principles / Mozilla Developer Network [Електронний ресурс] – URL: https://developer.mozilla.org/en-US/docs/Glossary/REST.

37 Single Page Application (SPA) Architecture and Routing Overview [Електронний ресурс] – URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA.

38 Vercel Cloud Deployment and Hosting Documentation [Електронний ресурс] – URL: https://vercel.com/docs/.

39 Vite Next Generation Frontend Tooling Documentation [Електронний ресурс] – URL: https://vite.dev/.

40 Visual Studio Code Editor Architecture and Extensions Documentation [Електронний ресурс] – URL: https://code.visualstudio.com/docs.

41 Kharchenko O., Bodnarchuk I., Galay I. Trade-off for quality attributes of software architecture on the base of multicriterion choice models // The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM) : Proceedings of the XII International Conference. — Lviv-Polyana : IEEE, 2013. — P. 145–147.

42 Bodnarchuk I., Kharchenko O., Galay I. Multicriteria architecture choice of software system under design and reengineering // The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM) : Proceedings of the XIII International Conference. — Lviv-Polyana : IEEE, 2015. — P. 328–330.

43 Боднарчук І. О., Галай І. О. Метод багатокритеріальної оптимізації програмної архітектури на основі аналізу компромісів // Інженерія програмного забезпечення. — Київ : НАУ, 2012. — № 3–4 (11–12). — С. 28–37.

44 Харченко О. Г., Боднарчук І. О., Яцишин В. В. Модель якості та метод багатокритеріального оцінювання архітектури програмних систем // Вісник Національного університету «Львівська політехніка». Серія: Інформаційні системи та мережі. — Львів : Вид-во Львівської політехніки, 2011.

ДОДАТКИ

Додаток А

Відомість програмного комплексу клієнтської частини веб-сайту

Лістинг А.1 – Вихідний код головного компонента управління та маршрутизації React Single Page Application

```
import { Suspense, lazy } from "react";
import { Route, Routes } from "react-router-dom";
import Loader from "./components/Loader/Loader.jsx";
import Header from "./components/Header/Header.jsx";

const HomePage = lazy(() =>
import("./pages/HomePage/HomePage.jsx"));
const CatalogPage = lazy(() =>
import("./pages/CatalogPage/CatalogPage.jsx"));
const AiToolDetailsPage = lazy(() =>
  import("./pages/AiToolDetailsPage/AiToolDetailsPage.jsx")
);
const FavoritePage = lazy(() =>
  import("./pages/FavoritePage/FavoritePage.jsx")
);
const NotFoundPage = lazy(() =>
  import("./pages/NotFoundPage/NotFoundPage.jsx")
);

const App = () => {
  return (
    <>
      <Header />
      <main>
        <Suspense fallback={<Loader />}>
          <Routes>
            <Route path="/" element={<HomePage />}></Route>
            <Route path="/catalog" element={<CatalogPage
/>}></Route>
            <Route
              path="/catalog/:id/*"
              element={<AiToolDetailsPage />}
            ></Route>
            <Route path="/favorites" element={<FavoritePage />} />
            <Route path="*" element={<NotFoundPage />}></Route>
          </Routes>
        </Suspense>
      </main>
    </>
  );
};

export default App;
```

Лістинг А.2 – Вихідний код логіки взаємодії з клієнтським сховищем LocalStorage та управління станом обраного

```
import { useEffect, useState } from "react";
import { useLocation } from "react-router-dom";
import { useDispatch, useSelector } from "react-redux";

import Loader from "../Loader/Loader.jsx";
import ErrorMessage from "../ErrorMessage/ErrorMessage.jsx";
import AiTool from "../AiTool/AiTool.jsx";

import { selectAiTools } from "../../redux/aiTools/selectors.js";
import { selectError, selectLoading } from
"../../redux/global/selectors.js";
import { fetchAiTools } from "../../redux/aiTools/operation.js";
import { selectFavorites } from
"../../redux/favorite/selectors.js";

import css from "./AiToolsList.module.css";

const AiToolsList = () => {
  const [page, setPage] = useState(1);
  const perPage = 4;
  const { pathname } = useLocation();
  const dispatch = useDispatch();

  const favoriteAiToolsIdArray = useSelector(selectFavorites);
  const aiTools = useSelector(selectAiTools);
  const loading = useSelector(selectLoading);
  const error = useSelector(selectError);

  useEffect(() => {
    dispatch(fetchAiTools());
  }, [dispatch]);

  const favoriteAiTools = aiTools?.filter((aiTool) =>
    favoriteAiToolsIdArray?.includes(aiTool._id),
  );

  const whichPageIsIt = pathname === "/catalog" ? aiTools :
favoriteAiTools;
  const totalPages = Math.ceil(whichPageIsIt?.length / perPage);

  const getCurrentPageData = () => {
    const start = (page - 1) * perPage;
    const end = start + perPage;
    return whichPageIsIt?.slice(0, end);
  };

  const handleClick = () => {
    if (page < totalPages) {
      setPage((prevPage) => prevPage + 1);
```

```
    }
  };

  return (
    <>
      {loading && <Loader />}
      {error && <ErrorMessage error={error} />}

      {!error &&
        !loading &&
        getCurrentPageData() &&
        getCurrentPageData()?.length > 0 && (
          <div>
            <ul className={css.aiToolsList}>
              {Array.isArray(getCurrentPageData()) &&
                getCurrentPageData().map((aiTool) => {
                  return <AiTool key={aiTool._id} aiTool={aiTool}
/>;
                })}
            </ul>
            {page < totalPages && (
              <button className={css.loadMoreBtn}
onClick={handleClick}>
                Load more
              </button>
            )}
          </div>
        )}

      {!error && !loading && getCurrentPageData()?.length === 0 &&
(
        <div className={css.nothingToShow}>Nothing to show</div>
      )}
    </>
  );
};

export default AiToolsList;
```

Додаток Б

Відомість програмного комплексу серверної частини веб-сайту

Лістинг Б.1 – Вихідний код головного файлу ініціалізації веб-сервера Express.js та підключення бази даних

```
import express from "express";
import cors from "cors";
import dotenv from "dotenv";
import mongoose from "mongoose";
import { Tool } from "./models/Tool.js";

dotenv.config();

const app = express();
const PORT = process.env.PORT || 5000;

app.use(cors());
app.use(express.json());

mongoose
  .connect(process.env.MONGODB_URI)
  .then(() => console.log("Успішно підключено до MongoDB Atlas"))
  .catch((err) => console.error("Помилка підключення до БД:",
err));

app.get("/api/ai-tools", async (req, res) => {
  try {
    const filters = {};

    if (req.query.developer) {
      filters.developer = new RegExp(req.query.developer, "i");
    }

    if (req.query.category) {
      filters.category = req.query.category;
    }

    if (req.query.hasApi === "true") {
      filters.hasApi = true;
    }

    if (req.query.hasMobileApp === "true") {
      filters.hasMobileApp = true;
    }

    if (req.query.hasVision === "true") {
      filters.hasVision = true;
    }
```

```
    if (req.query.hasVoice === "true") {
      filters.hasVoice = true;
    }

    if (req.query.accessType) {
      filters.accessType = req.query.accessType;
    }

    const tools = await Tool.find(filters);
    res.json({ items: tools });
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

app.get("/api/ai-tools/:id", async (req, res) => {
  try {
    const tool = await Tool.findById(req.params.id);
    if (!tool) return res.status(404).json({ message: "Not found"
});
    res.json(tool);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

app.listen(PORT, () => {
  console.log(`Сервер успішно запущений на порту: ${PORT}`);
});
```

Лістинг Б.2 – Вихідний код схеми документів NoSQL бази даних MongoDB на основі Mongoose ODM

```
import { Schema, model } from "mongoose";

const reviewSchema = new Schema({
  reviewer_name: { type: String, required: true },
  reviewer_rating: { type: Number, required: true },
  comment: { type: String, required: true },
});

const toolSchema = new Schema({
  name: { type: String, required: true },
  developer: { type: String, required: true },
  rating: { type: Number, required: true },
  description: { type: String, required: true },
  category: { type: String, required: true },
  accessType: { type: String, required: true },
  interfaceType: { type: String, required: true },
  licenseType: { type: String, required: true },
  hasApi: { type: Boolean, default: false },
  hasMobileApp: { type: Boolean, default: false },
```

```
  hasPluginSystem: { type: Boolean, default: false },
  hasVision: { type: Boolean, default: false },
  hasVoice: { type: Boolean, default: false },
  gallery: [
    {
      original: { type: String, required: true },
    },
  ],
  releaseYear: { type: String },
  contextWindow: { type: String },
  legalStatus: { type: String },
  pricingDetails: { type: String },
  reviews: [reviewSchema],
});

export const Tool = model("Tool", toolSchema);
```