

Міністерство освіти і науки України
Державний заклад
«Луганський національний університет імені Тараса Шевченка»

Навчально-науковий інститут математики та інформаційних технологій

Кафедра інформаційних технологій та систем


Карлов Максим Олегович

**ЦИФРОВІЗАЦІЯ АГРАРНИХ ПРОЦЕСІВ ЧЕРЕЗ СТВОРЕННЯ
ПОВНОФУНКЦІОНАЛЬНОГО ВЕБДОДАТКУ (MERN-СТЕК)**

кваліфікаційна робота

**здобувача вищої освіти другого (магістерського) рівня
освітньої програми «Мультимедійні системи»
за спеціальністю 121 „Інженерія програмного забезпечення”**

Особистий підпис



Максим КАРЛОВ

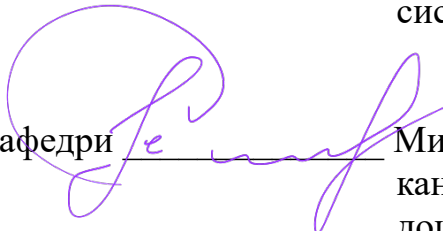
Науковий керівник



Галина КОЗУБ,

кандидат технічних наук, доцент
кафедри інформаційних технологій та
систем

Завідувач кафедри



Микола СЕМЕНОВ,

кандидат педагогічних наук,
доцент кафедри інформаційних технологій
та систем

Лубни – 2026

АНОТАЦІЯ

Тема: Цифровізація аграрних процесів через створення повнофункціонального вебдодатку (MERN-стек)

Спеціальність: 121 «Інженерія програмного забезпечення».

Установа: ДЗ ЛНУ імені Тараса Шевченка, 2026р.

Магістерська робота містить: 86 стор., 1 таблиця, 14 рисунків, 56 джерел

Об'єкт дослідження: ІС обліку тварин

Предмет дослідження: Технології вебпрограмування HTML, CSS, JS, React, Redux та MongoDB

Мета роботи: Розробка вебдодатку обліку тварин у господарстві та аналіз існуючих інформаційних систем.

Результати роботи: У результаті проведеного дослідження розроблено веб-додаток обліку тварин у господарстві, який дозволяє автоматизувати процес ведення інформації про поголів'я. Створено нову інформаційну систему, орієнтовану на зручну взаємодію користувача з базою даних тварин, що дозволяє підвищити точність, швидкість обліку та зменшити людський фактор. Система реалізована за допомогою сучасних технологій веб-програмування: HTML, CSS, JavaScript, React, Redux, Node.js та MongoDB. Також враховано потенціал інтеграції IoT-рішень (RFID, сенсори, GPS) для майбутнього розширення функціональності системи. Впровадження такої інформаційної системи сприяє підвищенню ефективності управління тваринницьким господарством та є кроком до цифрової трансформації агросектору.

Ключові слова: ВЕБДОДАТОК, MERN, NODE, EXPRESS, HTML, CSS, JS, REACT, REDUX, MONGODB

ANNOTATION

Topic: Digitalization of agricultural processes through the creation of a fully functional web application (MERN stack)

Specialty: 121 "Software Engineering".

Institution: Taras Shevchenko National University of Lviv, 2026.

The master's thesis contains: 86 pages., 1 tables, 14 images, 56 references

Research object: Animal registration system

Subject of research: Web programming technologies HTML, CSS, JS, React, Redux and MongoDB

Purpose of work: Development of a web application for recording animals on the farm and analysis of existing information systems.

Results of the work: As a result of the research, a web application for animal accounting on the farm was developed, which allows you to automate the process of maintaining information about the livestock. A new information system was created, focused on convenient user interaction with the animal database, which allows you to increase the accuracy, speed of accounting and reduce the human factor. The system was implemented using modern web programming technologies: HTML, CSS, JavaScript, React, Redux, Node.js and MongoDB.

The potential for integrating IoT solutions (RFID, sensors, GPS) for the future expansion of the system's functionality was also taken into account. The implementation of such an information system contributes to increasing the efficiency of livestock management and is a step towards the digital transformation of the agricultural sector.

Keywords: WEB APPLICATION, MERN, NODE, EXPRESS, HTML, CSS, JS, REACT, REDUX, MONGODB

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ІС	Інформаційна система
MERN	MongoDB, Express.js, React.js, Node.js
MEAN	MongoDB, Express.js, Angular, Node.js
MEVN	MongoDB, Express.js, Vue.js, Node.js
LAMP	Linux, Apache, MySQL, PHP/Python/Perl
WAMP	Windows, Apache, MySQL, PHP
WIMP	Windows, IIS, MySQL, PHP
CRUD	Create, Read, Update, Delete
REST API	Representational State Transfer API
JSON	JavaScript Object Notation
BSON	Binary JSON
ODM	Object Data Modelling
ORM	Object-Relational Mapping
UI	User Interface
UX	User Experience
SPA	Single Page Application
CSR	Client-Side Rendering
SSR	Server-Side Rendering
SSG	Static Site Generation
ISR	Incremental Static Regeneration
API	Application Programming Interface
IoT	Internet of Things
RFID	Radio Frequency Identification
GPS	Global Positioning System
HTTP	HyperText Transfer Protocol

URL	Uniform Resource Locator
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
Node.js	Середовище виконання JavaScript
Express.js	Фреймворк для Node.js
React.js	JavaScript-бібліотека
Redux	Бібліотека керування станом
NPM	Node Package Manager
JWT	JSON Web Token
BBS	Bulletin Board System
CGI	Common Gateway Interface
AJAX	Asynchronous JavaScript and XML
PWA	Progressive Web Application
ACID	Atomicity, Consistency, Isolation, Durability
TLS/SSL	Transport Layer Security / Secure Sockets Layer
SCRAM	Salted Challenge Response Authentication Mechanism
LDAP	Lightweight Directory Access Protocol
CMS	Content Management System
IIS	Internet Information Services
Nginx	Високопродуктивний вебсервер
Apache	HTTP-сервер
Mongoose	ODM-бібліотека для MongoDB

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1. ТЕОРЕТИЧНІ ЗАСАДИ ТЕХНОЛОГІЙ СТВОРЕННЯ ВЕБДОДАТКІВ	13
1.2 Еволюція технологій розробки вебдодатків	21
1.3 MERN стек: огляд технологій MongoDB, Express.js, React.js, Node.js	31
1.3.1 Архітектура MERN стеку	33
1.3.1.1 Фронтенд React.js	34
1.3.1.2 Серверний рівень Express.js і Node.js.....	34
1.3.1.3 Рівень бази даних MongoDB	34
1.3.1.4 Приклад запиту/відповіді в MERN.....	35
1.3.1.5 Коли обрати MERN?	36
1.3.2 Переваги MERN	36
1.3.3 Застосування MERN	37
1.3.3.1 MongoDB: особливості та взаємодія	37
1.3.3.2 Express.js: створення сервера та обробка HTTP-запитів.....	40
1.4 Висновки до 1 розділу.....	43
РОЗДІЛ 2. РОЗРОБКА МОДАЛЬНОГО ІНТЕРФЕЙСУ ВВЕДЕННЯ ОБ'ЄКТІВ У КЛІЄНТ-СЕРВЕРНОМУ ЗАСТОСУНКУ ДЛЯ АГРАРНОГО КОМПЛЕКСУ	45
2.1 Концепція та завдання розробки.....	46
2.2 Архітектура інтеграції модального інтерфейсу	46
2.3 Реалізація модального інтерфейсу введення даних	48
2.4 Графічний вигляд і принципи UX-дизайну модального компонента ..	49
2.5 Переваги реалізованого рішення	51
2.6 Висновки до розділу 2.....	51
РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБДОДАТКУ ОБЛІКУ ТВАРИН НА ОСНОВІ MERN-СТЕКУ.....	52
3.1. Загальна архітектура системи	52

3.2 Загальна структура Frontend.....	52
3.2.1 Компонентна архітектура	53
3.2.2 Сторінки та маршрутизація	54
3.2.3 Взаємодія з сервером.....	54
3.3 Загальна структура Backend	54
3.4 Реалізація та архітектурні особливості серверної частини вебдодатку.....	55
3.4.1. Призначення серверного рівня системи.....	55
3.4.2. Технологічна основа бекенд-частини.....	56
3.4.3. Організація серверного проєкту	56
3.4.4. Опис моделі обліку тварин	57
3.4.5. Побудова програмного інтерфейсу взаємодії	57
3.4.6. Контроль коректності та обробка помилок	57
3.4.7. Особливості взаємодії з клієнтським інтерфейсом.....	57
3.4.8. Узагальнення результатів реалізації бекенду	58
3.5. Висновки до розділу 3.....	58
ВИСНОВКИ.....	60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	63
ДОДАТКИ	70
Додаток 1. Лістинг коду списку кроликів(Rabbits.jsx).....	70
Додаток 2. Лістинг коду модального вікна додавання кролика(AddRabbitModal.jsx).....	72
Додаток 3. Лістинг коду авторизації користувача (Register.jsx).....	75
Додаток 4.Лістинг коду взаємодії з бекенд сервером	77
Додаток 5. Лістинг коду взаємодії з бекенд сервером(feedOperation.js)	79
Додаток 6. Лістинг коду бекенду Mongoose-схеми (Rabbits.js)	81
Додаток 7. Лістинг коду бекенду модель-схеми (../models/Rabbits.js)	82
Додаток 8. Лістинг коду бекенду контролера.....	84
Додаток 9. Сертифікат	86

ВСТУП

Актуальність теми. Цифровізація аграрного сектору. Сучасне сільське господарство активно переходить до цифрових технологій. У тваринництві це означає автоматизоване управління процесами, точний облік та ефективний моніторинг. Інформаційні системи стають критично важливими для підвищення ефективності виробництва.

У сучасному агропромисловому комплексі інформаційні технології відіграють ключову роль у підвищенні ефективності виробництва. Особливо актуальним є впровадження інформаційних систем у тваринництві — галузі, яка потребує точного обліку, контролю й аналізу великої кількості даних щодо утримання, годування, здоров'я та продуктивності тварин.

Зниження рентабельності традиційного тваринництва. В умовах зростання вартості кормів, енергоносіїв та трудових ресурсів, тваринницькі підприємства шукають способи оптимізації витрат. Інформаційні системи дозволяють зменшити витрати, оптимізувати ресурси та підвищити продуктивність.

Потреба в точному управлінні та аналітиці. Тваринництво вимагає оперативного реагування на зміни в стані тварин. ІС дозволяють збирати великі обсяги даних (про здоров'я, поведінку, споживання корму, надої), обробляти їх і приймати обґрунтовані рішення.

Дефіцит кваліфікованих кадрів. Завдяки інформаційним системам знижується залежність від людського фактора. Сучасні платформи можуть автоматично вести облік, контролювати параметри утримання та надавати рекомендації.

Попит на екологічно чисту та якісну продукцію. Системи моніторингу дозволяють контролювати умови утримання, уникати антибіотиків і підвищувати якість продукції — що є актуальним для вимог споживача. Це

визначає актуальність і значущість проблеми якій присвяченні дослідження розробки вебзастосунку для аграрних процесів.

Мета дослідження — аналіз і дослідження сучасних вебтехнологій та розробка повнофункціонального вебзастосунку для обліку тварин у господарстві “Rabbits” з використанням MERN-стеку з метою автоматизації облікових процесів і підвищення ефективності управління інформацією.

Об’єкт дослідження — інформаційні системи (ІС) обліку тварин.

Предмет дослідження — методи, засоби та технології вебпрограмування, що використовуються для проєктування і реалізації клієнт-серверних інформаційних систем обліку тварин, зокрема застосування HTML, CSS, JavaScript, бібліотеки React, менеджера стану Redux, серверної платформи Node.js та документоорієнтованої бази даних MongoDB.

Гіпотеза дослідження є припущення, що впровадження клієнт-серверного вебзастосунку, розробленого з використанням сучасних вебтехнологій та MERN-стеку, забезпечить підвищення ефективності обліку тварин у господарстві за рахунок автоматизації процесів обробки даних, зменшення впливу людського фактора та покращення доступності інформації.

Відповідно до мети дослідження сформульовані такі завдання:

- Проаналізувати сучасний стан та методологічні підходи до створення і впровадження вебзастосунків у сфері аграрних систем.
- Дослідити архітектуру та особливості компонентів MERN-стеку (MongoDB, Express.js, React.js, Node.js).
- Розробити структуру клієнт-серверного вебдодатку для автоматизації обліку тварин у господарстві.
- Реалізувати модуль введення даних у вигляді модального інтерфейсу з використанням технологій React і REST API.
- Провести тестування функціональних модулів системи та оцінити ефективність її роботи.

– Сформулювати рекомендації щодо подальшого вдосконалення вебсистем аграрного призначення.

Методи дослідження. У процесі виконання магістерської роботи використано комплекс загальнонаукових і спеціальних методів дослідження, що забезпечили досягнення поставленої мети та розв’язання визначених завдань.

Методи аналізу та синтезу застосовувалися для вивчення наукових джерел, нормативної документації та існуючих інформаційних систем у сфері тваринництва з метою узагальнення підходів до цифровізації аграрних процесів. Порівняльний аналіз використовувався для оцінки функціональних можливостей наявних систем обліку тварин та вибору оптимальних технологічних рішень.

Особистий внесок здобувача. У межах виконання роботи проведено аналіз сучасного стану цифровізації аграрних процесів, зокрема у сфері тваринництва; здійснено огляд і порівняльний аналіз існуючих інформаційних систем обліку тварин; обґрунтовано вибір архітектури програмного продукту та технологічного стеку MERN (MongoDB, Express.js, React.js, Node.js) з урахуванням функціональних вимог, масштабованості та можливостей подальшого розвитку системи; спроектовано структуру бази даних, визначено логіку взаємодії між клієнтською та серверною частинами, а також реалізовано REST API для обробки запитів та управління даними; розроблено користувацький інтерфейс вебзастосунку, протестовано працездатність програмного продукту. Усі етапи розробки, тестування та документування результатів виконані здобувачем особисто. Консультації наукового керівника мали рекомендаційний характер і не впливали на самостійність виконання магістерської роботи.

Апробація і впровадження результатів магістерської роботи. Апробація результатів магістерської роботи здійснювалась під час розробки, налагодження та тестування вебзастосунку для обліку тварин у господарстві, реалізованого з використанням MERN-стеку. Матеріали роботи доповідалися на VII Міжнародній науково-практичній конференції *“Ricerche scientifiche e metodi della loro realizzazione: esperienza mondiale e realta domestiche”*, яка відбулася у червні 2025р., м. Болонья. За результатами конференції надруковано статтю за темою “Розробка модального інтерфейсу введення об'єктів у клієнт-серверному застосунку для аграрного комплексу”.[1] У ході перевірки оцінювалась працездатність функціональних модулів, коректність обробки даних та взаємодія клієнтської і серверної частин системи. Практичні напрацювання можуть бути використані як основа для подальшого впровадження цифрових рішень у сфері тваринництва та розвитку інформаційних систем аграрного призначення.

Структура магістерської роботи. Робота складається з текстового документу та програмного продукту. Текстова частина включає вступ, три розділи, висновки, список використаних джерел та додатки.

У першому розділі проведено аналіз теоретичних основ технологій розробки веб додатків, розглянуто генезу вебтехнологій та особливості використання MERN-стеку. Автор детально описав архітектуру стеку, його переваги та сфери застосування.

У другому розділі представлено аналіз сучасних інформаційних систем у тваринництві, таких як DairyComp, PigCHAMP, DelPro, SmartBarn. Висвітлено актуальність цифровізації аграрного сектору, окреслено проблеми та переваги впровадження інформаційних систем у господарствах.

У третьому розділі описано процес розробки веб додатку обліку тварин у господарстві. Робота містить опис використаних технологій (HTML, CSS, JavaScript, React, Redux, Node.js, MongoDB), структуру клієнтської та серверної

частини, а також можливості інтеграції IoT-рішень (RFID, сенсори, GPS) для майбутнього розширення функціоналу.

Додатки містять фрагменти коду розробленого застосунку та сертифікат.

РОЗДІЛ 1

ТЕОРЕТИЧНІ ЗАСАДИ ТЕХНОЛОГІЙ СТВОРЕННЯ ВЕБДОДАТКІВ

Глобальна комп'ютерна мережа Інтернет упродовж останніх десятиліть стала невід'ємною складовою майже всіх сфер людської діяльності. З часу її відкритого використання минуло понад тридцять років, і за цей період вона докорінно трансформувала повсякденне життя, професійну діяльність та суспільні процеси. На сучасному етапі Інтернет виступає ключовим середовищем для обміну інформацією, розвитку бізнесу, реалізації соціальних ініціатив, а також функціонування різноманітних інформаційно-організаційних систем. Значна кількість компаній постійно розширює можливості власних вебресурсів, орієнтуючись на потреби широкої аудиторії користувачів.

Станом на 2025 рік кількість користувачів Інтернету у світі перевищила приблизно 5,56 мільярда осіб, що становить близько 67,9 % населення планети. З них 5,24 мільярда людей активно користуються соціальними мережами та іншими цифровими платформами. [54]

Найвищі показники проникнення Інтернету у 2025 році були зафіксовані у країнах Північної та Західної Європи, де доступ до мережі мають практично всі мешканці (близько 97–99 %). У той же час у деяких регіонах Африки рівень доступу залишається значно нижчим за середньосвітовий показник. За абсолютною кількістю користувачів Інтернету лідерами залишаються Китай, Індія та Сполучені Штати Америки. [55]

В Україні рівень проникнення Інтернету також демонструє зростання: на початок 2025 року його показник сягнув близько 82,4 %, а кількість інтернет-користувачів становила приблизно 31,5 мільйона осіб. Це відображає стабільну інтеграцію населення до цифрового середовища. [56]

Об'єднуючи мільярди користувачів у різних країнах, Інтернет сьогодні становить фундамент інформаційного суспільства. Його вплив за масштабами можна порівняти з ключовими технологічними революціями в історії людства.

Поява та розвиток таких ресурсів, як Google, Amazon чи YouTube, суттєво змінили способи отримання інформації, споживання товарів і взаємодії між людьми (рис. 1.1).

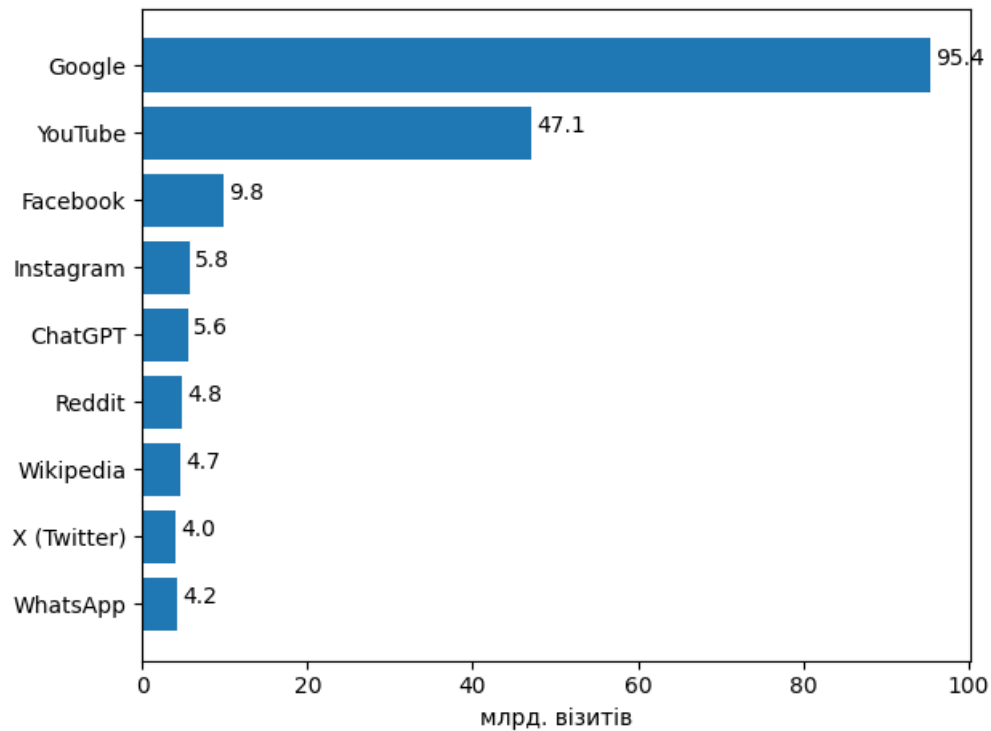


Рис.1.1. Найпопулярніші вебсайти за кількістю відвідувань у 2025 році

За даними відповідних досліджень, у 2025 році загальна кількість вебресурсів у глобальній мережі перевищила понад 1,2 мільярда сайтів, хоча лише частина з них (близько 16–18 %) є активними та регулярними у оновленні контенту. [54]

Зростання залежності суспільства від Інтернету зумовлює підвищені вимоги до швидкодії, стабільності, зручності та доступності вебдодатків. Сучасні вебсистеми зазвичай являють собою складні програмні комплекси, що інтегруються з базами даних, платіжними сервісами та іншими інформаційними платформами.

1.1 Сутність вебдодатків та їхня роль у сучасному світі.

Алан Кей, один із засновників об'єктно-орієнтованого програмування, влучно зауважив, що «технологія – це те, чого не існувало на момент вашого народження». Цікаво відзначити, що термін «вебсайт» з'явився в українському законодавстві лише у 2017 році, тоді як «вебдодаток» досі не має офіційного визначення в нормативно-правових актах. Згідно з чинним законодавством, «веб-сайт» трактується як організована сукупність даних, електронної (цифрової) інформації, об'єктів авторського та/або суміжних прав, взаємопов'язаних і структурованих у межах певної веб-адреси, доступ до яких забезпечується через мережу Інтернет. У тому ж законі «веб-сторінка» визначається як складова частина веб-сайту, що розміщена за конкретною адресою в Інтернеті.

У 2019 році оновлена редакція Українського правопису внесла зміни до правил написання деяких слів, зокрема термінів, що починаються з компонента «веб». Відтепер такі слова, як «вебсайт», «вебресурс», «вебзастосунок», «вебсторінка», «вебдодаток», пишуться разом, без дефісу. Хоча процес узгодження чинних законів та нормативних актів з новим правописом відбувається повільно, він все ж триває. Це призводить до тимчасових розбіжностей в українському науковому середовищі, і вітчизняні дослідники, вивчаючи «веб»-тематику, стикаються з певними термінологічними невідповідностями.

Міжнародна наукова спільнота, яка є рушійною силою сучасних ідей та технологічним лідером, активно використовує термін «web application». Слово «application» традиційно перекладалося українською як «програма», «додаток» або «застосунок». З появою терміна «web application» виникли його українські відповідники – «вебдодаток» або «вебзастосунок». Таким чином, при вивченні вебдодатків важливо враховувати весь діапазон синонімів цього терміна, а також їхні попередні граматичні форми.

На сучасному етапі дослідження вебдодатків привертають дедалі більшу увагу як зарубіжних, так і вітчизняних науковців. Зокрема, порівняльний аналіз ефективності фреймворків та бібліотек, що використовуються для розробки вебдодатків, є предметом досліджень S. Skrzypiec, M. Plechawska-Wójcik, K. Bielak та B. Borek. [9,10]

Архітектура вебдодатків, створених за допомогою фреймворку Angular, детально вивчена V. Garcia. Автор узагальнив свій практичний досвід розробника та запропонував ефективні рішення для створення трьох повноцінних програм.[11]

Дослідження C. Rojas, Z. Tahir, A. Ilham, M. Niswar, A. Fauzy, B. Susanto, B. Virginia, G. Proboyekti та U. Ester присвячені розробці прогресивних вебдодатків.[12,13,14]

Праця M. Baker зосереджена на актуальній проблемі захисту вебдодатків від кібератак. Автор акцентує увагу на методах, якими хакери здійснюють атаки на вебдодатки, та описує широкий спектр засобів захисту. Д. Бартлетт, розробник програмного забезпечення та дослідник, у своїй роботі приділяє значну увагу створенню вебдодатків на базі PHP та представляє кілька хмарних рішень, що сприяють оптимізації їхньої роботи.[15]

Впровадженню машинного навчання для реалізації потенціалу штучного інтелекту присвячена робота V. Karunanidhi. Автор розробив покрокову систему інструкцій для розгортання моделі машинного навчання у вебдодатку. Зокрема, у своїй праці дослідник використовує бібліотеку TensorFlow (JavaScript) для розгортання таких моделей. [17]

У статті M. Слабіноги та С. Чабан [18] автори зосередили увагу на розробці рекомендацій для проєктування вебдодатків, що забезпечують максимально швидке відображення користувацького інтерфейсу, а також на створенні вебдодатків з урахуванням цих рекомендацій. Дослідники здійснили аналіз предметної області та підходів до проєктування вебдодатків, визначивши критерії їхньої ефективності. Розроблені авторами вдосконалені методи

проектування веборієнтованого програмного забезпечення, що враховують рекомендації щодо підвищення продуктивності вебдодатків, дозволяють досягти вищої швидкодії.

В. Зосімов вивчав методи видобування даних, а також моделі та підходи до створення мов обробки даних для корпоративних вебресурсів та інтеграції семантичної розмітки в HTML-код. Автор використовує термін «веб-ресурс» як синонім вебсайту або вебдодатку. Після детального аналізу інформаційного наповнення та навігаційних елементів 1000 корпоративних вебресурсів українського сегменту Інтернету, дослідник розробив онтологію семантичних класів та властивостей для опису їхнього інформаційного вмісту. На цій основі він створив абстрактну модель системи комплексного управління даними в мережі Інтернет, а також програмний комплекс для її реалізації. [19]

О. Войтюк та Д. Плечистий у своїй роботі розглядають процеси оптимізації рендерингу вебзастосунків із застосуванням об'єктів з глибокою вкладеністю. За допомогою браузерних інструментів автори дослідили проблеми ефективності під час промальовування вебдодатків. [20]

Стаття О. Кошової та співавторів присвячена всебічному дослідженню особливостей розробки вебдодатків для систем дистанційного навчання з використанням бібліотеки React. Авторка висвітлила переваги та особливості застосування бібліотеки React у процесі розробки вебдодатків [21].

Проблему вибору конструктивних елементів для створення вебдодатків за допомогою CMS-систем розглядали В. Федько та Б. Безуглий. Дослідники виявили ключові недоліки використання CMS-технологій під час розробки та запропонували власну систему, засновану на колекції програмних хуків та віджетів. Ця система прискорює розробку, спрощує взаємодію з базою даних та сприяє швидкій реалізації спільного функціоналу. [23]

Варто зазначити, що в українському науковому середовищі досі тривають дискусії щодо точного визначення поняття «вебдодаток». Наприклад, «Тлумачний словник з інформатики» за авторством Г. Півняка [22] та інших не

містить визначень для «вебдодатку» чи «вебзастосунку». Проте в ньому наведено поняття «веб-застосування (web application)», яке описується як «набір клієнтських і серверних програм, що функціонують спільно для вирішення завдань обробки контенту, розміщеного на веб-серверах...». Інший варіант визначення цього терміна, запропонований авторами словника, трактує вебзастосунок як сукупність вебсторінок, що відображаються у браузері.

У виданні «Базові поняття і терміни веб-технологій» [24] серед ключових термінів представлено «вебзастосунок». Автори визначають його як «клієнт-серверний застосунок, де браузер виступає клієнтом, а веб-сервер – сервером. Логіка вебзастосунку розподілена між сервером і клієнтом, а зберігання даних переважно здійснюється на сервері, обмін інформацією відбувається через мережу».

С. Майстренко та співавтори у своїй статті, присвяченій технологіям зберігання та візуалізації даних, визначають вебдодаток як «клієнт-серверний додаток, де браузер виступає клієнтом, а веб-сервер – сервером». Автори також дійшли висновку, що вебдодатки, як правило, складаються з трьох ключових компонентів: сервера, клієнта та бази даних. [25]

С. Ужейко розглядає вебдодаток як «клієнтську програму, яка дозволяє користувачеві вирішувати прикладні завдання. На відміну від традиційних застосунків, вебзастосунок зазвичай включає серверну частину, що виконується на віддаленому веб-сервері, та клієнтську частину, яка функціонує безпосередньо у браузері на пристрої користувача». Дослідник підкреслює незалежність вебдодатків від операційної системи та версій встановленого програмного забезпечення, що забезпечує їхню доступність з будь-якого пристрою. [27]

Н. Матвеева та В. Нусс визначають вебдодаток як «комплекс, що є сукупністю програмного забезпечення, сервера та системного коду. Це дозволяє розробляти унікальні веб-сторінки для конкретних завдань». [26]

В. Щербань, описуючи компоненти веб-технологій, визначає вебдодаток як програмне забезпечення, доступ до якого можливий через будь-який браузер. На думку науковця, відмінність вебдодатків від вебсайтів полягає в тому, що вони відображають не лише статичні дані (тексти, зображення), а й динамічні програми. Однак, на наш погляд, сучасні вебдодатки, створені на базі фреймворків, здатні відтворювати й статичні дані, і в такому випадку, згідно з цим визначенням, вони можуть бути класифіковані як вебсайти. [28]

У публічному просторі вебдодатки іноді характеризуються як «динамічний вебсайт, що надає користувачам графічний інтерфейс...», або як «програма, що слугує допоміжним засобом для виконання певних операцій на веб-серверах», або навіть як «сукупність статичних і динамічних вебсторінок, тобто заздалегідь створених сторінок і програм, які генерують вебсторінки у відповідь на запити користувача». Нерідко вебсайт і вебдодаток розглядаються як тотожні поняття.

Інтернет-гігант Amazon, що є найбільшою у світі платформою хмарних обчислень, визначає вебдодаток як «програмне забезпечення, що функціонує у браузері» [32]. Пояснюючи сутність вебдодатків, Amazon зазначає, що після появи Інтернету вебсайти мали значно менший функціонал порівняно з сучасними вебдодатками. Вони могли надавати користувачам лише статичну інформацію. Веб-додатки були розроблені для подолання обмежень статичних сайтів. З того часу веб-технології значно еволюціонували, і сьогодні більшість сучасних вебсайтів за своєю архітектурою є складними вебдодатками.

Щодо поняття «вебдодаток», думки міжнародної спільноти також різняться. D. Brooks [29] зауважував, що нині термін «вебдодаток» став загальновживаним, а інтерактивність є важливою складовою більшості вебсайтів, оскільки вона зробила їх складнішими, але водночас цікавішими для розробки.

L. Baresi, [30] досліджуючи проблеми концептуального проектування вебдодатків, пропонує розглядати їх як розширення традиційної інформаційної

системи, доповнене навігацією та складними інформаційними структурами. Або ж як розширення традиційних вебсайтів, збагачених різними видами операцій. А. Belfrage [31] вважав, що вебдодаток – це вебсайт, орієнтований на виконання конкретного завдання, наприклад, онлайн-каталог, інтернет-банкінг або веб-система навігації.

А. Хоффман, [34] аналізуючи архітектуру, безпеку та механізми захисту сучасних вебдодатків від хакерських атак, визначає вебдодаток як програму, що завантажується та виконується у браузері. На його думку, вебдодатки

від традиційних вебсайтів наявністю багаторівневої системи дозволів, можливістю зберігати введені користувачем дані в базах даних та часто дозволяють користувачам обмінюватися контентом між собою.

С. Фаулер та В. Стенвік [33] у своїй праці, намагаючись відповісти на питання «що таке вебдодаток і чим він відрізняється від сайту», дійшли висновку, що межа між цими поняттями є дуже розмитою, і вебсайти та вебдодатки слід розглядати як континуум без чітких розмежувань.

У роботі, присвяченій веброзробці на Node та Express, Е. Браун [35] зазначає, що «сайт – це вебдодаток, а вебсторінка – це також вебдодаток». Автор вважає, що слово «додаток» загалом використовується для позначення будь-якого функціонального елемента, що не є просто статичним набором контенту, хоча й статичний контент він називає дуже простим прикладом вебдодатків. Дослідник також прогнозує, що за кілька років відмінності між вебдодатком та вебсайтом повністю зникнуть.

Отже, аналіз наявної наукової літератури дозволяє констатувати, що термін «вебдодаток» досі не має остаточного, чіткого визначення як в українській, так і в міжнародній науковій спільноті. На наш погляд, найбільш вдале та узагальнене формулювання цього поняття запропонувала професорка Інституту інформаційних систем Віденського технічного університету G. Kappel [36]. За її визначенням, вебдодаток – це програмна система, створена на основі технологій та стандартів World Wide Web Consortium (W3C), яка через

веббраузер надає специфічні вебресурси, включаючи контент та сервіси. З цим визначенням важко не погодитися, оскільки W3C є основним джерелом усіх технологій та стандартів, на яких базується Інтернет. Саме ця організація сприяє довгостроковому розвитку Інтернету шляхом розробки прогресивних протоколів та правил. Однак важливо зазначити, що на сьогоднішній день під це визначення також підпадає поняття «вебсайт», оскільки він також є системою, заснованою на технологіях та стандартах W3C. Визначення відмінностей між вебсайтом та вебдодатком іноді є складним завданням. Наразі можна стверджувати, що ці поняття, якщо не тотожні, то дуже близькі між собою.

1.2 Еволюція технологій розробки вебдодатків

На сучасному етапі розвитку інформаційних технологій, методи створення вебдодатків посідають центральне місце, відіграючи ключову роль у цифровій трансформації. Історія розвитку технологій вебдодатків є складною мозаїкою інновацій, стандартів та методологій, що пройшла шлях від простих статичних веб-сторінок до високопродуктивних інтерактивних застосунків. Аналіз ключових етапів та тенденцій у цій галузі дозволяє виявити основні чинники, що формують архітектурну парадигму сучасної веброзробки, а також спрогнозувати її майбутні напрямки.

Існування вебдодатків неможливе без веб-браузерів, оскільки їхня функціональність реалізується саме в їхньому середовищі. Тому початком історії вебдодатків можна вважати створення першого браузера. У 1989 році в Європейській організації з ядерних досліджень (CERN) у Швейцарії англійський програміст і фізик Тім Бернерс-Лі представив керівництву дві пропозиції щодо майбутньої мережі Інтернет. Хоча жодна з них не отримала схвалення, він продовжив роботу і до кінця 1990 року розробив прототип «WorldWideWeb». Проєкт Бернерса-Лі включав сервер, мову HTML, URL-адреси та перший браузер під назвою Nexus. Цей браузер виконував функції як переглядача, так і редактора (текстового процесора), підключеного до Інтернету,

що відображало початкове бачення автора щодо інтеграції інструментів розробки в саму мережу.

Варто зазначити, що ідеї Т. Бернерса-Лі мали попередників. Зокрема, Т. Нельсон та Д. Енгельбарт ще у 1950-х роках запропонували концепцію комп'ютеризованих перехресних посилань, які стали основою для гіперпосилань, що використовуються в сучасному Інтернеті. Нельсон назвав це «гіперпосиланням», а комп'ютеризований текст – «гіпертекстом».

Наприкінці 1970-х років у США спостерігалось поступове зростання кількості персональних комп'ютерів та їхнє підключення до перших онлайн-сервісів, таких як електронні дошки оголошень (BBS), Usenet та мережі обміну файлами (BITNET). До 1990 року понад два мільйони мешканців Північної Америки вже користувалися онлайн-сервісами для участі в дискусійних групах, здійснення покупок, отримання новин, спілкування та електронної пошти.

У 1980 році Т. Бернерс-Лі в CERN розробив систему «Enquire» – мережеву гіпертекстову систему, призначену для управління проектами, але з набагато ширшими амбіціями. Він прагнув створити гіперпосилання таким чином, щоб їх могли інтерпретувати як комп'ютери, так і люди. Пізніше він зазначав, що на той час не був обізнаний з ранніми роботами Нельсона та Енгельбарта з гіпертексту, тому його розробка могла бути незалежним переосмисленням.

У той період ідея створення глобальної мережі була широко підтримана, проте не було єдиної думки щодо її реалізації. До початку 1980-х років існувало кілька конкуруючих національних та корпоративних мережевих протоколів. Першим офіційним стандартом з міжнародною підтримкою стала модель OSI (Open Systems Interconnect), офіційно опублікована у 1984 році. Ця модель дозволила різним пристроям взаємодіяти один з одним через мережу.

Саме в таких умовах був створений проєкт Т. Бернерса-Лі. Його основна ідея полягала у можливості «зв'язувати» різні типи дослідницьких документів. Робота Бернерса-Лі та його колег заклала основи нового протоколу передачі

гіпертексту – HTTP, та нової мови гіпертекстової розмітки – HTML. Сьогодні HTTP, хоча й абстрагований від безпосереднього сприйняття користувачем, залишається фундаментальною основою наших додатків.

Після винайдення Т. Бернерсом-Лі, інші представники наукової спільноти почали використовувати Інтернет для індексування своїх наукових документів. Однак оригінальний браузер-редактор Бернерса-Лі працював лише на рідкісних комп'ютерах NeXT, а CERN відмовився фінансувати розробку версій для інших платформ. У відповідь на це, команда розробила простий, текстовий браузер для швидкого розповсюдження, надала його код і закликала волонтерів адаптувати його для широкого кола комп'ютерів. Це призвело до появи кількох браузерів, таких як Viola, Midas та Mosaic, який згодом перетворився на Netscape Navigator. З того часу Т. Бернерс-Лі втратив прямий контроль над своїм творінням.

Варто зазначити, що виробники браузерів, прагнучи покращити свої продукти, почали додавати власні набори тегів до HTML-розмітки. Це призвело до фрагментації: різні HTML-документи могли коректно відображатися лише в певних браузерах. Розробникам доводилося створювати кілька версій HTML-документів для кожної окремої програми перегляду. У 1993 році розширений стандарт HTML+ додав нові функції, такі як зображення, таблиці та форми, а у 1995 році вийшов новий стандарт HTML 2.0.

Найбільш популярним браузером на початку 1990-х років став Mosaic. Він дозволив інтегрувати зображення безпосередньо в мову розмітки. Ця інновація значно пожвавила раніше сухі, перевантажені текстом веб-сторінки. До цього зображення позначалися як посилання і відкривалися в окремому вікні після натискання.

На основі коду Mosaic, Марк Андрессен створив перший комерційний браузер – Netscape Navigator, який швидко здобув визнання та стрімко поширювався. Саме для Netscape у 1995 році Брендан Айк розробив мову JavaScript, без якої сучасний Інтернет важко уявити. Netscape також представив світу іншу важливу технологію – cookies (куки), невеликі блоки даних, які

сервер зберігає на комп'ютері користувача для збереження інформації або відстеження дій. Наприклад, у cookies можна зберігати товари, додані до кошика в онлайн-магазині, і вони залишаться там, коли користувач повернеться на сайт. Cookies з'явилися у 1994 році і вже за рік були прийняті іншими браузерами. Довгий час cookies не мали альтернатив, проте згодом браузери додали інші API для збереження стану додатків, такі як LocalStorage, SessionStorage, Web SQL, IndexedDB тощо.

Водночас, коли Netscape домінував на ринку, компанія Microsoft, використовуючи код Mosaic, створила свій перший Internet Explorer. Завдяки популярності операційної системи Windows, в яку був вбудований браузер від Microsoft, до 1998 року популярність Netscape та ІЕ спочатку зрівнялася, а потім почалося багаторічне домінування Microsoft на ринку браузерів.

Конкуренція між браузерами призвела до появи нестандартних функцій у кожному окремому браузері. Найбільші розбіжності виникли у підтримці JavaScript – мови сценаріїв, що забезпечує інтерактивність документів. В результаті багато документів були «оптимізовані» для конкретного браузера і не відтворювалися коректно в інших. Хоча W3C приймав численні детально обговорені стандарти, відповідальність за їх дотримання все ж лягала на розробників браузерів.

Користувачі прагнули більшої динаміки, а виробники браузерів намагалися забезпечити багатший клієнтський досвід. Першими з'явилися Java-аплети – попередньо скомпільований байткод Java, який завантажувався в браузер і виконувався. Ця технологія випередила свій час, але через різні причини, пов'язані з безпекою та продуктивністю, не набула широкої популярності.

З 1996 року Microsoft розробляла Active Server Pages (ASP) – власний механізм для створення динамічних веб-сторінок. Технологія ASP використовувала сценарії на сервері для генерації контенту, який надсилався до веб-браузера клієнта через HTTP. Ці сценарії писалися мовами VBScript, JScript

або PerlScript, а для запитів до сервера використовувалися об'єкти MSXML2.ServerXMLHTTP та Microsoft.XMLHTTP. Згодом ця концепція була реалізована в Internet Explorer 5 за допомогою ActiveXObject("Msxml2.XMLHTTP"). Інші браузерери для асинхронних запитів до сервера використовували JavaScript-клас XMLHttpRequest, який з 2002 року підтримувався більшістю з них. У 2006 році його було додано до специфікації, а з 2012 року XMLHttpRequest почав підтримувати і Internet Explorer. До появи цієї технології HTML-форма вимагала повного оновлення сторінки браузера після відправки на сервер. Новий метод спілкування веб-сторінки з сервером отримав назву AJAX (Asynchronous JavaScript and XML).

Паралельно з цим, з 1996 року технологія Flash стала популярним методом для створення анімації та інтерактивності на сайтах. За допомогою Flash створювалися рекламні банери, відео, інтерактивні елементи дизайну і навіть повноцінні веб-сайти. Технологія стрімко розвивалася, проте протягом багатьох років експерти наголошували на проблемах безпеки Flash. У 2015 році виробники браузерів Firefox, Chrome та Safari внесли Flash Player до чорного списку, що призвело до його поступового зникнення. Одним із перших веб-серверів, розроблених після сервера Т. Бернерса-Лі, був сервер NCSA, випущений у 1993 році. До 1995 року NCSA обслуговував більшість веб-серверів в Інтернеті, проте майже всі вони швидко перейшли на Apache. У квітні 1996 року Apache випередив NCSA і був провідним сервером в Інтернеті до середини 2016 року. Наступні три роки лідерство належало IIS від компанії Microsoft, але зрештою і він втратив позиції завдяки Nginx – високопродуктивному веб-серверу з відкритим кодом.

Оскільки дизайн Всесвітньої павутини за своєю суттю не був динамічним, ранній гіпертекст складався з вручну закодованих текстових HTML-файлів, які публікувалися на веб-серверах. У 1993 році для взаємодії сайтів із веб-серверами був представлений стандарт Common Gateway Interface (CGI). Він вперше забезпечив динамічність даних веб-сторінок. Цей інтерфейс дозволив

веб-сторінкам викликати серверні програми, написані мовами C, Perl та іншими, а також обробляти дані, надіслані зі сторінки. Досить швидко з'явилися альтернативи CGI, такі як FastCGI, SCGI, модулі Apache тощо, які використовуються й досі.

Перші скрипти для динамічних сайтів були написані мовами C та Perl. Після найпростіших скриптів, що обробляли дані з форм, з'явилися чати, форуми та інші динамічні сайти. У 1995 році з'явилася мова PHP. Її історія почалася, коли автор Расмус Лердорф, після появи інтерфейсу CGI, написав кілька програм на C для підтримки форм на своєму персональному сайті. Він зазначав: «Я не знаю, як це зупинити. Ніколи не було наміру писати мову програмування. Я абсолютно не знаю, як писати мови програмування». Станом на травень 2025 року 74,2% веб-сайтів використовують саме PHP (рис. 1.2).

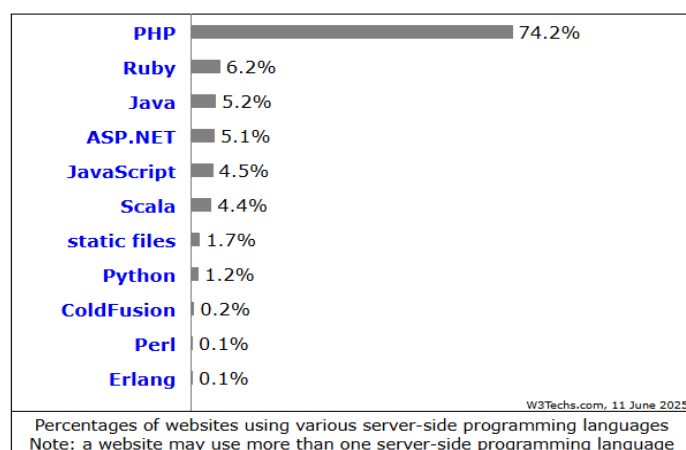


Рис. 1.2. Відсоток сайтів, що використовують ту чи іншу серверну мову

У 1995 році на ринку з'явилося рішення від компанії FileNet для управління документами на сервері, яке вважається першим справжнім рішенням для управління контентом. Після цього з'явилися й інші корпоративні CMS, серед яких найвідоміші: Interwoven (1995), Documentum (1996), FatWire (1996), Future Tense (1996), Inso (1996), EPiServer (1997).

У 1998 році М. Кунце продемонстрував спільноті набір безкоштовного програмного забезпечення з відкритим кодом, який міг стати реальною альтернативою дорогим комерційним пакетам. До цього набору входили Linux як операційна система, Apache як HTTP-сервер, MySQL як база даних, а також

PHP, Perl або Python як мови програмування. Так виник один із перших стеків технологій розробки програмного забезпечення – LAMP. Згодом ця модель була адаптована для використання з іншими компонентами. Наприклад, зміна операційної системи на Windows дає стек WAMP, а зміна сервера на IIS – WIMP. Наразі існує багато варіацій. Стек LAMP і сьогодні активно використовується у розробці вебдодатків.

Для спрощення та прискорення веброзробки з початку 2000-х років почали з'являтися так звані «фреймворки» – комплексні рішення, що забезпечували стандартизований спосіб створення та розгортання сайтів в Інтернеті. Серед найбільш популярних: Laravel, Ruby on Rails, Symfony, Spring, Yii, Django, Flask. Також з'явилися перші повноцінні CMS: Drupal, WordPress, Joomla та інші.

У 2009 році Р. Даль представив світу Node.js – середовище виконання JavaScript з відкритим вихідним кодом, що працює в Windows, Linux, Unix, macOS та інших системах. Ця нова платформа дозволила розробникам використовувати JavaScript не лише у браузері, а й для створення сценаріїв на стороні сервера. Невдовзі Node.js отримала менеджер пакетів NPM, який дозволив програмістам публікувати та вільно обмінюватися програмними пакетами.

Платформа Node.js виявилася надзвичайно продуктивною. У Node.js використовуються три ключові концепції: події, асинхронний API та неблокуючий ввід/вивід. Це означає, що програма може відправити запит до зовнішнього ресурсу, зайнятися іншими завданнями, а потім, після завершення зовнішньої асинхронної операції, виконати функцію зворотного виклику, яка обробляє результат.

Завдяки появі Node.js виник ще один стек технологій – MEAN (MongoDB – база даних, Express.js – сервер, Angular – фреймворк для створення користувацького інтерфейсу та Node.js – платформа). Всі компоненти стеку MEAN підтримують код, написаний на JavaScript. Таким чином, тепер можна

однією мовою писати як для серверних, так і для клієнтських середовищ виконання. Як і у випадку зі стеком LAMP, тут також з'явилися варіанти з іншими компонентами. Варіант, відомий як MERN, замінює Angular бібліотекою React.js, а інший під назвою MEVN використовує Vue.js.

Автор Node.js Р. Даль розробив ще одну платформу, наступницю Node.js під назвою Deno. Проте наразі вона ще не отримала широкого визнання і потребує доопрацювання. Іншим кандидатом на заміну Node є нова розробка під назвою Bun. Її автор Дж. Самнер обіцяє кращу швидкість та повну зворотну підтримку коду, що вже написаний під Node. Проєкт активно розвивається, і станом на вересень 2024 року вийшов стабільний реліз 1.1.29.

З розвитком JavaScript для покращення візуальної складової веб-сайтів з'явилося багато бібліотек. Однією з найпопулярніших стала бібліотека jQuery, призначена для маніпулювання DOM-деревом веб-сторінки, а також обробки подій, анімації та AJAX. Це найпоширеніша бібліотека JavaScript. Станом на 2024 рік jQuery використовують 77% із 10 мільйонів найпопулярніших веб-сайтів.

Прагнення до подальшого покращення користувацького інтерфейсу та прискорення його роботи призвело до появи низки бібліотек та фреймворків, що дозволяли створювати сайти за архітектурою SPA (Single Page Application). При такому підході веб-браузер завантажує лише одну сторінку, а потім динамічно оновлює її новими даними з сервера. При цьому повністю імітується поведінка звичайного веб-сайту. За потреби, за технологією AJAX динамічно підвантажуються додаткові ресурси. Найпопулярнішими рішеннями в цьому сегменті наразі є React, Angular, Vue та Svelte. Підхід відтворення веб-сторінок безпосередньо в браузері за допомогою JavaScript отримав назву CSR, або Client-Side Rendering.

Логічним продовженням цієї технології стала можливість збереження отриманих файлів на пристрої користувача, оскільки вони вже були завантажені. Так з'явилися PWA – Progressive Web Application. PWA дозволили користуватися

додатком без підключення до Інтернету, а з отриманням доступу до мережі автоматично синхронізуватися з сервером. PWA наразі стали популярним засобом для відправлення так званих Push-повідомлень, які сайт може надсилати користувачу в будь-який час.

Зростання популярності напрямку SPA виявило неочікуваний ефект. Пошукові системи не могли ефективно індексувати контент, що генерується односторінковими сайтами, оскільки сервер віддавав фактично порожній HTML. Контент додавався вже згодом за допомогою JavaScript. Проте трафік з пошукових систем є дуже важливою складовою сучасного інтернет-бізнесу. Як рішення цієї проблеми виникла низка технологій, що дозволяють частково чи повністю генерувати веб-сторінки на сервері: SSR (Server-Side Rendering), SSG (Static Site Generation) та ISR (Incremental Static Regeneration).

У випадку з SSR, для обчислень на сервері потрібен більш потужний сервер, ніж у випадку з CSR, де всі обчислення відбувалися на клієнтській машині.

Static Site Generation (SSG) – це рішення, яке виявилось корисним у випадках, коли сайт є досить невеликим або оновлюється нечасто. У такому разі немає необхідності щоразу генерувати сторінку на сервері. Можна заздалегідь згенерувати певну кількість статичних HTML-файлів і відправити їх браузеру. Завдяки цьому клієнт максимально швидко отримує свій файл, оскільки немає потреби його генерувати. Відповідно, знижується навантаження на сервер. Недоліком є те, що у разі оновлення будь-яких даних потрібно провести повне перерендеринг всього сайту, що може стати проблемою, якщо сайт містить сотні тисяч сторінок. Додатковою перевагою є покращення безпеки веб-сайту, оскільки клієнт отримує лише HTML, а також економія на ресурсах веб-хостингу. У випадку потреби в динамічних сервісах зазвичай використовуються Serverless-рішення від хмарних провайдерів, такі як AWS Lambda або Cloud Functions.

Incremental Static Regeneration (ISR) – це рішення, що поєднує методи SSR та SSG. У цьому випадку частина веб-сторінок, що часто оновлюються, генерується динамічно, а частина тих, що практично не оновлюються, генеруються статично. Серед недоліків цього підходу можна відзначити лише складність реалізації. Проте існуючі фреймворки значно допомагають у цьому, адже кожен з лідерів розробки SPA наразі має технології для SSR: Next.js для React, Angular Universal для Angular та Nuxt.js для Vue. Існують також і незалежні рішення.

Розглядаючи сьогоденне розмаїття технологій для створення веборієнтованих рішень, стає зрозумілою складність спроб описати чіткі критерії, за якими можна було б однозначно визначити, чи є перед вами сайт, чи вебдодаток. Тим більше, що визначення терміну «вебдодаток» ще остаточно не прийняте науковою спільнотою.

Проте дослідження еволюції цих технологій, на наш погляд, чітко показало кілька етапів, або рівнів, які щоразу давали поштовх новому розвитку веброботки. Для опису цих рівнів дуже зручно використовувати модель зрілості (рис. 1.3), що охоплює шлях від початкового рівня – перших статичних HTML-сторінок – до сучасних складних програмних рішень.

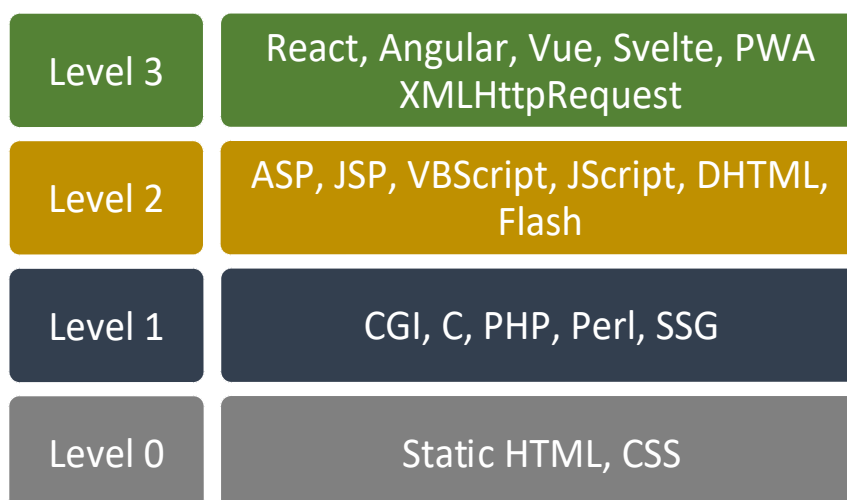


Рис. 1.3. Рівні зрілості веб додатків

На нульовому рівні знаходяться, як вже було зазначено, статичні сторінки. Тобто ті рішення, що створені за допомогою чистого, статичного HTML, ми пропонуємо називати «веб-сайтами» або вебдодатками нульового рівня зрілості.

Перший рівень займають вебдодатки, що використовують або створені за допомогою лише серверних бекенд-технологій. Ці вебдодатки вже можуть обробляти запити з браузера, динамічно генерувати сторінки. Проте вони відправляють браузеру той самий статичний HTML-код. Прикладами є перші поштові форми, генератори статичних сторінок, а також деякі вебдодатки стеку LAMP.

На другому рівні зрілості з'являються технології, що вже мають певну динаміку на стороні веб-браузера. Такі вебдодатки вже можуть маніпулювати DOM-деревом, дозволяють з клієнтської сторони робити запити до сервера, використовуючи вбудовані рішення.

На найвищому на сьогодні рівні розміщені технології і, відповідно, створені за їх допомогою вебдодатки, що тим чи іншим способом використовують у роботі браузера клас XMLHttpRequest або його спадкоємця – Fetch API. Це всі рішення, що використовують AJAX, а також сучасні бібліотеки та фреймворки, які допомагають у створенні вебдодатків.

Відповідність конкретного вебдодатка рівню зрілості визначається за максимальним рівнем використаної в ньому технології.

На наш погляд, описана модель є зручним і доступним способом пояснити важливі відмінності вебдодатків та різницю між вебдодатком і сайтом. Також наведена модель дозволяє послідовно зрозуміти основні етапи розвитку технологій веброзробки.

1.3 MERN стек: огляд технологій MongoDB, Express.js, React.js, Node.js

MERN є варіантом технологічного стеку MEAN (MongoDB, Express, Angular, Node), де Angular.js замінено на React.js. Існують також інші подібні

конфігурації, наприклад MEVN (MongoDB, Express, Vue, Node), а також інші JavaScript-фреймворки для фронтенду.

Перед тим, як детально розглядати MERN, варто зрозуміти, що таке технологічний стек. Це набір технологій, які обирають для розробки веб- або мобільних застосунків. Ефективний стек має забезпечувати безперебійний користувацький досвід, масштабованість і економічність. Зазвичай стек включає фронтенд, бекенд і базу даних, формуючи повний технологічний стек (рис. 1.4).

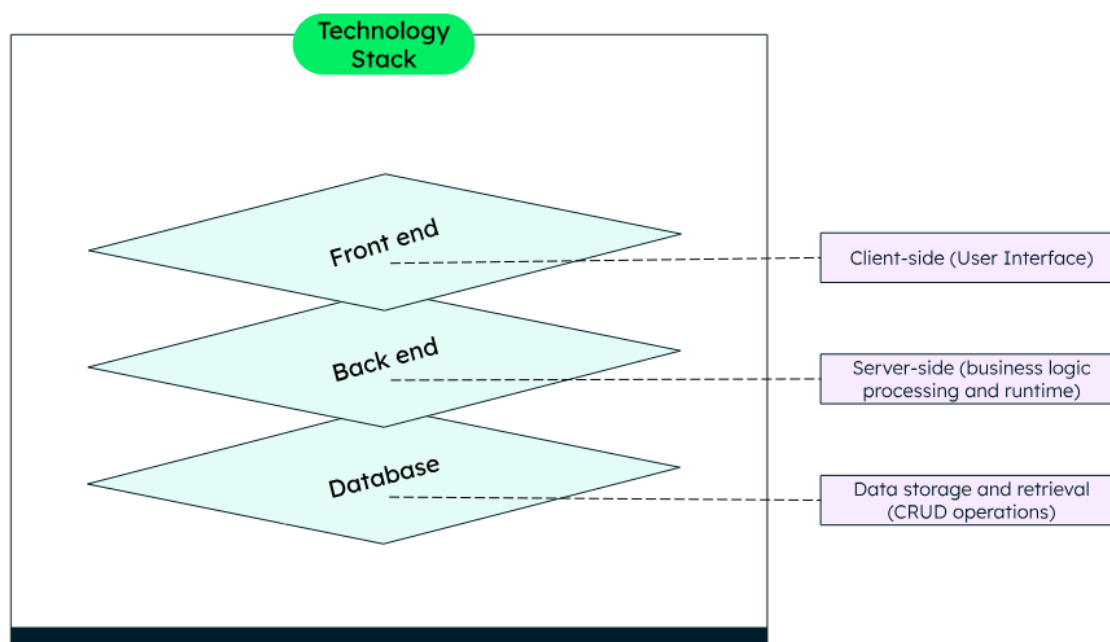


Рис. 1.4 Повний технологічний стек

Для фронтенду зазвичай використовують HTML, CSS і JavaScript. Залежно від проєкту, застосовують бібліотеки або фреймворки, такі як React або Angular, які базуються на цих технологіях.

Бекенд — це серверна частина, де реалізується логіка програми. Для цього використовують різні мови програмування (JavaScript, Java, Python) та фреймворки (Django, Spring, Express.js). Для виконання коду потрібне середовище, наприклад Node.js або JRE.

База даних зберігає всі необхідні дані. Вона може бути реляційною (таблична структура) або нереляційною (NoSQL), наприклад документною чи графовою. Вибір бази залежить від структури даних. Прикладами є MongoDB, Oracle, MySQL.

Технологічний стек може бути індивідуальним (підбирається під проєкт) або попередньо визначеним.

MERN — це попередньо сформований стек на основі JavaScript. Назва MERN розшифровується як MongoDB, Express, React і Node, що позначає ключові компоненти:

- MongoDB — документоорієнтована база даних.
- Express.js — веб-фреймворк для Node.js.
- React.js — клієнтський JavaScript-фреймворк.
- Node.js — середовище виконання JavaScript на сервері.

Express і Node формують серверний рівень. Express.js — це серверний веб-фреймворк, а Node.js — популярна серверна платформа на JavaScript. ME(RVA)N стек ідеально підходить для роботи з JavaScript і JSON.

1.3.1 Архітектура MERN стеку

На рисунку 1.5 показана архітектура MERN:

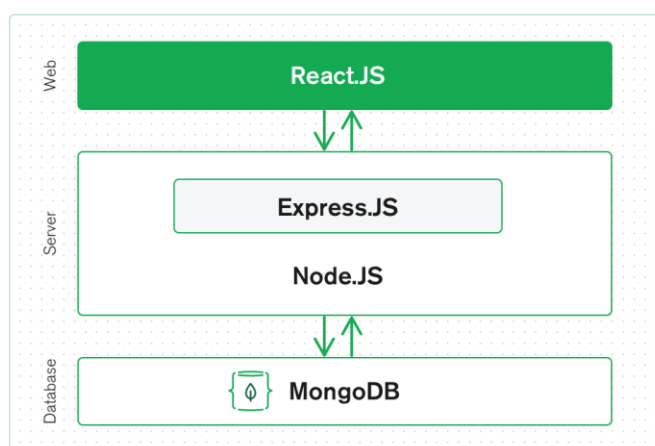


Рис. 1.5 Робота MERN-стеку

Ця архітектура дозволяє створювати трирівневі застосунки (фронтенд, бекенд, база даних), повністю використовуючи JavaScript і JSON.

1.3.1.1 Фронтенд React.js

Верхній рівень MERN — це React.js, декларативний JavaScript-фреймворк для створення динамічних клієнтських інтерфейсів у HTML. React дозволяє будувати складні інтерфейси з простих компонентів, інтегрувати їх із серверними даними та відображати у вигляді HTML.

React ефективно обробляє інтерфейси, керовані даними, з мінімальним кодом. Він підтримує форми, обробку помилок, подій, списків та інші функції сучасного веб-фреймворку.

1.3.1.2 Серверний рівень Express.js і Node.js

Наступний рівень — серверний фреймворк Express.js, що працює в середовищі Node.js. Express.js — це швидкий, мінімалістичний веб-фреймворк для Node.js, який забезпечує маршрутизацію URL та обробку HTTP-запитів і відповідей.

HTTP-запити з фронтенду React.js (XHR, GET, POST) взаємодіють з функціями Express.js, які реалізують логіку програми. Ці функції через драйвери Node.js працюють з базою даних MongoDB.

1.3.1.3 Рівень бази даних MongoDB

Якщо застосунок потребує зберігання даних (профілі користувачів, контент, коментарі тощо), потрібна база даних, з якою так само просто працювати, як із React, Express і Node.js.

MongoDB дозволяє зберігати JSON-документи, які надсилаються з React.js на сервер Express.js, де вони обробляються і зберігаються в MongoDB. Для хмарних застосунків є MongoDB Atlas. Для налаштування власного MERN стеку читайте далі.

1.3.1.4 Приклад запиту/відповіді в MERN

HTTP-запити (POST, GET, PUT, DELETE) відповідають операціям CRUD (створення, читання, оновлення, видалення) у базі даних. Express.js обробляє об'єкти `request` і `response`, які містять параметри запиту і дані відповіді.

За допомогою драйвера MongoDB Node.js можна легко підключити MongoDB до програми.

Важливо, що всі технології MERN працюють з даними в одному форматі: React зберігає дані як JavaScript-об'єкти, бекенд використовує JavaScript, а MongoDB зберігає BSON (бінарний JSON). Express конвертує дані між JS і JSON через `.json()`.

Node.js дозволяє писати серверну логіку на JavaScript. Express додає структуру і маршрутизацію, що спрощує роботу з HTTP-запитами.

Наприклад, клієнт хоче оновити номер телефону через форму React. React зберігає номер у стані. Express отримує PUT-запит, перевіряє дані і оновлює запис у MongoDB через метод `findByIdAndUpdate()`. Відповідь надсилається клієнту (рис. 1.6-1.7).

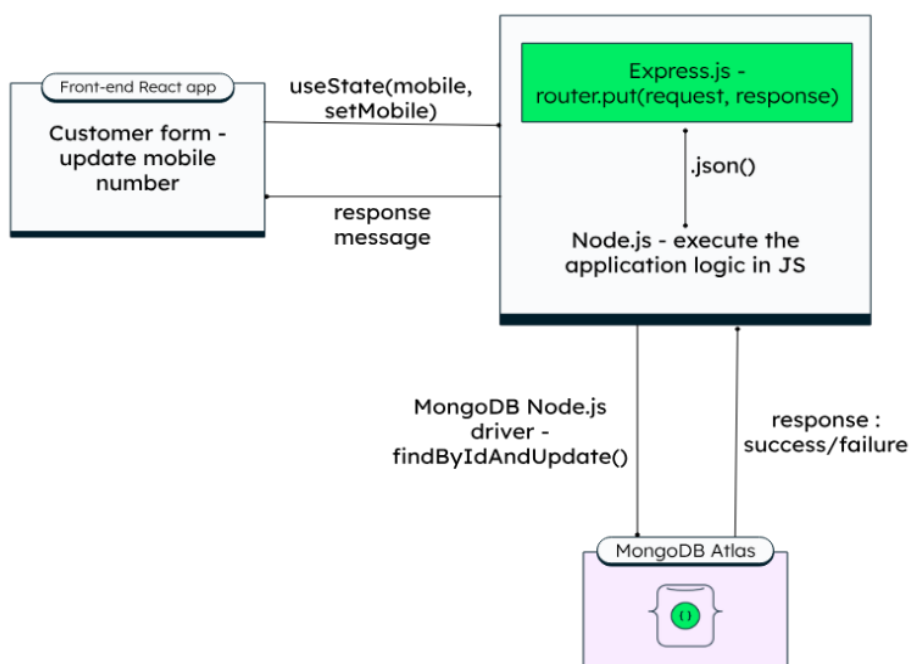


Рис. 1.6 Приклад функції `findByIdAndUpdate()`

MERN стек і повний стек розробки

MERN — це різновид повного стеку з визначеними технологіями. Розробник повного стеку має знати кілька технологій, а MERN-розробник — лише MERN-компоненти. Порівняння допоможе вибрати між користувацьким повним стеком і MERN (рис. 1.7).

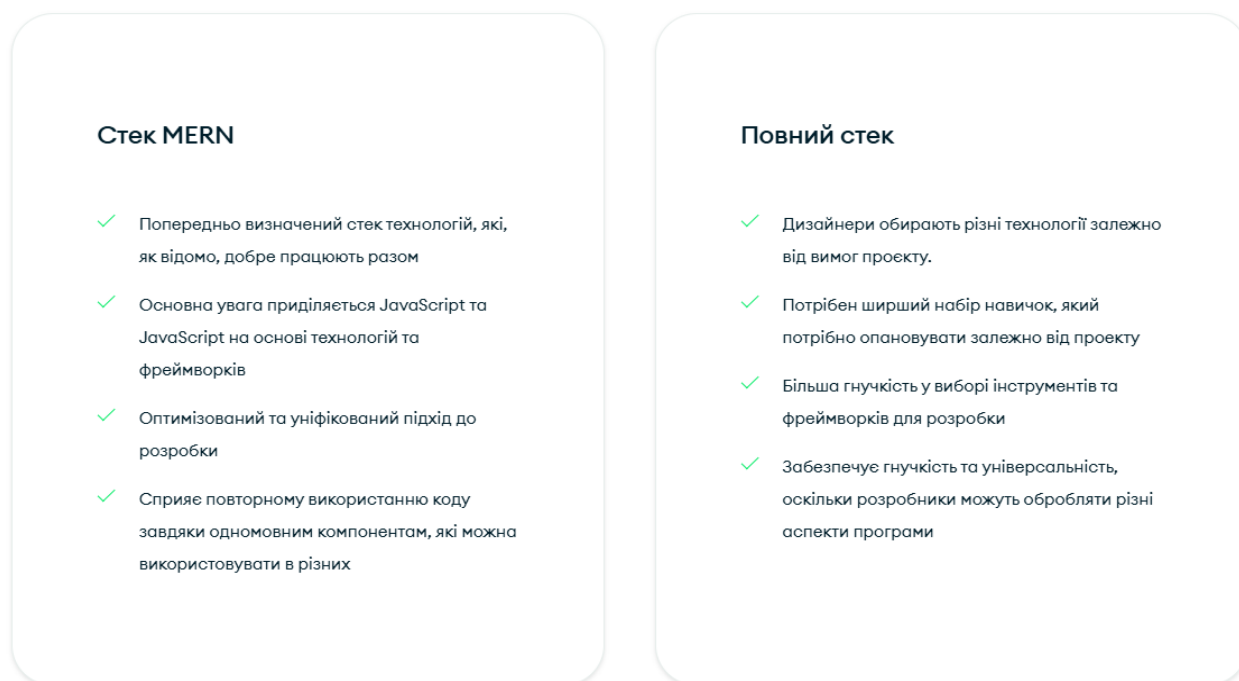


Рис. 1.7 Порівняння стеків

1.3.1.5 Коли обрати MERN?

Якщо проєкт має обмежені терміни і чіткі вимоги, MERN — оптимальний вибір. Він економить час і кошти, а розробникам потрібно вивчати лише одну технологію. Обслуговування проєкту простіше завдяки структурованості і великій документації.

MERN підходить для будь-яких проєктів, де потрібен JavaScript. Виняток — випадки, коли потрібно інші технології.

1.3.2 Переваги MERN

MongoDB — основа MERN, документоорієнтована база, що зберігає дані у форматі BSON (бінарний JSON). Вона тісно інтегрується з Node.js, спрощуючи роботу з JSON на всіх рівнях.

MongoDB Atlas забезпечує хмарне масштабування.

Express.js і React.js роблять MERN повноцінним JavaScript/JSON-додатком. Express обробляє HTTP-запити, React створює інтерактивний інтерфейс.

Це забезпечує природний потік JSON-даних від початку до кінця, спрощуючи розробку і налагодження. Потрібна лише одна мова програмування.

MERN ідеальний для сучасних веб-розробників, особливо з досвідом React.js.

1.3.3 Застосування MERN

MERN підходить для різних застосунків, особливо тих, що базуються на JSON, хмарних і з динамічним веб-інтерфейсом: системи управління, новинні агрегатори, таск-менеджери, форуми, соціальні мережі тощо.

1.3.3.1 MongoDB: особливості та взаємодія

MongoDB створена у 2007 році командою, що працювала над DoubleClick, яка потребувала масштабованої бази даних. Зараз MongoDB підтримує масштабовані розподілені застосунки, має велику спільноту і багато клієнтів.

MongoDB зберігає дані у BSON, документи групуються в колекції, які утворюють базу даних. Підтримує різні типи даних, реплікацію, розділення даних, високу доступність і масштабованість.

MongoDB можна запускати локально або в хмарі через Atlas. Є корпоративні функції безпеки.

Нижче наведено порівняння MongoDB і MariaDB (таблиця 1.1).

Таблиця 1.1

Порівняльна таблиця MariaDB та MongoDB бази даних

	MariaDB	MongoDB
1	2	3
Модель даних	Зберігає дані в таблицях. Кожна таблиця містить рядки та стовпчики. Дані мають бути нормалізовані та поділені на різні логічні таблиці.	Зберігає документи в оптимізованому форматі BSON. Документи групуються в колекції та бази даних і повертаються як документи JSON із підтримкою великої кількості типів даних, включаючи: рядки, числа, геодані, дати, масиви, вкладені об'єкти, двійкові дані та ін.

Продовження таблиці 1.1

1	2	3
Індексація	Дозволяє індексувати різні стовпчики. Індокси охоплюють лише стовпчики, в яких вони створені. Якщо потрібні спеціальні індокси, наприклад геоіндекси, до бази даних потрібно додавати зовнішні модулі.	Підтримує багато типів індоксів для різних випадків використання. Вторинні індокси для будь-якого поля доступні та підтримуються в різних типах: текстовий, географічний, TTL та ін.
Мова запитів	використовує діалект SQL.	Має широкі можливості надсилання запитів, які надає Query API. Запити можуть використовувати складні оператори, а також структуру агрегації для розширеної обробки даних і аналітики.
Транзакції	Підтримує транзакції ACID, проте без підтримки snapshot isolation.	Підтримує повністю сумісні з ACID транзакції, включно з сегментованими кластерами та snapshot isolation.
Багато-поточність	Контроль багатопоточності в залежності від використаного механізму зберігання. Рівень ізоляції можна змінити на основі конкретних конфігурацій транзакцій. Дозволяє кільком користувачам бази даних одночасно отримувати доступ до тих самих даних, керуючи чітко визначеним контролем паралелізму.	Використовує блокування на рівні документа, тому записи в один документ відбуваються або повністю, або не відбуваються взагалі, і клієнти завжди бачать послідовні дані. Разом із цими механізмами підтримує читання та запис у розподілених кластерах.
Безпека	Автентифікація та авторизація через імена користувачів і паролі, TLS/SSL, шифрування x509, аудит.	Механізми безпеки корпоративного рівня. Автентифікація та авторизація за допомогою вбудованого SCRAM або сертифікатів, TLS/SSL, x509, шифрування з можливістю запиту та шифрування на рівні полів на стороні клієнта. Шифрування й аудит на стороні сервера. Інтеграції LDAP і Kerberos. Сертифікати відповідності вимогам безпеки.

Продовження таблиці 1.1

1	2	3
Мобільна розробка	Для мобільної розробки потрібно встановлювати додаткові сторонні модулі.	Два ключових компоненти для мобільної розробки: Atlas Device SDKs і Atlas Device Sync
Хмарні пропозиції	БД як послуга, частково доступна на AWS і GCP. Деякі функції доступні лише в GCP. Безкоштовний рівень недоступний. Вимагає попереднього визначення призначення кластера, наприклад транзакції чи аналітика. Обидва варіанти використання не можуть бути охоплені одним кластером.	БД як послуга доступна у трьох основних хмарних постачальників – AWS, Azure та GCP – починаючи від безкоштовного рівня до повномасштабного міжрегіонального та міжхмарного кластера. Послуги можна динамічно масштабувати, а хмарних провайдерів можна змінювати за потреби.
Документація та навчання	Пропонує документацію для версії з відкритим кодом і корпоративної версії. Також пропонує онлайн-курси та навчання.	Пропонує документацію з прикладами та повними посібниками. Має широку спільноту та вебсайти центру розробників. Є онлайн-університет із безкоштовними курсами.
Вбудовані інструменти візуалізації	Не пропонує жодних власних інструментів візуалізації даних, проте інтегрується зі сторонніми програмами для візуалізації даних і бізнес-аналітики.	MongoDB Charts забезпечує потужний спосіб візуалізації даних за допомогою MongoDB Atlas. MongoDB Compass надає клієнт GUI для MongoDB. Надає конектор для інтеграції з популярними сторонніми інструментами бізнес-аналітики.

MongoDB створена для хмарних технологій і роботи з великими обсягами даних, має потужний Query API і гнучку модель документів.

Для роботи з MongoDB часто використовують Mongoose — ODM-бібліотеку, що полегшує CRUD-операції, валідацію, міграції і запити.

1.3.3.2 Express.js: створення сервера та обробка HTTP-запитів

Express — це буква “E” в MEAN. Згідно даним Stack Overflow Express викликає неабиякий інтерес у розробників і останні роки знаходиться на першому місці по запитам на цьому сайті (рис. 1.8).

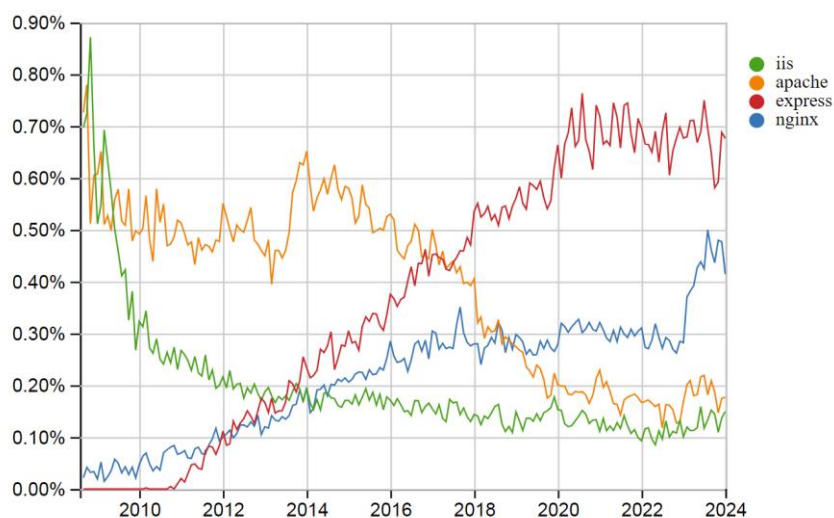


Рис. 1.8. Статистика трендів Stack Overflow по серверним технологіям[37]

Оскільки Node.js — платформа, вона потребує певних налаштувань. Express — це фреймворк, спроектований саме для спрощення цих налаштувань. Саме в ньому ми визначаємо які вхідні запити буде прослуховувати і обробляти вебсервер, які відповіді надсилати, структуру каталогів та ін. Фактично наш сервер це Node.js, а Express по суті — обгортка, допоміжний модуль.

В Node.js є вбудований модуль під назвою HTTP, який дозволяє передавати дані по протоколу передачі гіпертексту (HTTP). І якщо не використовувати Express, то для створення найпростішої точки доступу з відповіддю «Hello World!» достатньо набрати кілька строк коду:

Лістинг 1.1. Запуск серверу в Node.js

```
var http = require('http');
http.createServer(function (req, res) { // створюємо об'єкт сервера
  res.writeHead(200, {'Content-Type': 'text/html'});
```



```

    res.write('Hello World!'); // пишемо відповідь сервера
    res.end(); // кінець відповіді
  })
  .listen(3000); // об'єкт сервера слухає порт 3000

```

Запуск такого ж сервера за допомогою Express вийде ще елегантнішим:

Лістинг 1.2. Запуск серверу в Express.js

```

const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello World!');
});

app.listen(3000);

```

В якості альтернативи Express для роботи з сервером в Node використовуються також такі фреймворки як Koa, Sails, Nest та багато інших. Наприклад Sails більше буде до вподоби тим розробникам, що працювали з Ruby, а Nest тим, хто знайомий з Angular.

Для роботи з Express потрібно визначити роути (маршрути) та функції, що будуть спрацьовувати при зверненні клієнта за цими роутами (рис. 1.9).

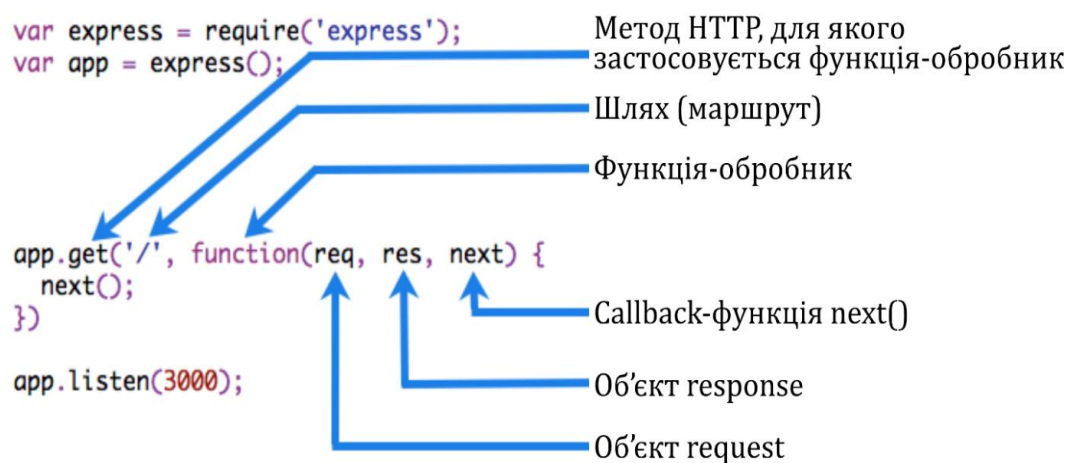


Рис. 1.9. Визначення роута в Express

На екземплярі сервера ми визиваємо метод HTTP, для якого застосовується функція-обробник. Першим аргументом вказуємо шлях

(маршрут), який ми будемо обробляти, Другим аргументом буде сама функція-обробник.

Вона приймає наступні аргументи:

- об'єкт `request`, що містить всі заголовки, методи, URL, тіло запиту;
- об'єкт `response`, в який ми додаємо те, що відправляється клієнту.

Цей об'єкт також містить кілька методів, що дозволяють генерувати різні відповіді в різних форматах;

- необов'язковий параметр `next`, для того щоб передати управління наступній функції-обробнику, якщо їх буде декілька.

В рамках функції-обробника ми виконуємо всі необхідні дії, що має зробити сервер при виклику за цією URL адресою в комбінації з цим HTTP методом. Фреймворк Express по суті – це набір таких роутів та обробників.

На практиці схема обробки запиту в Express може виглядати так як показано на рис. 1.10.

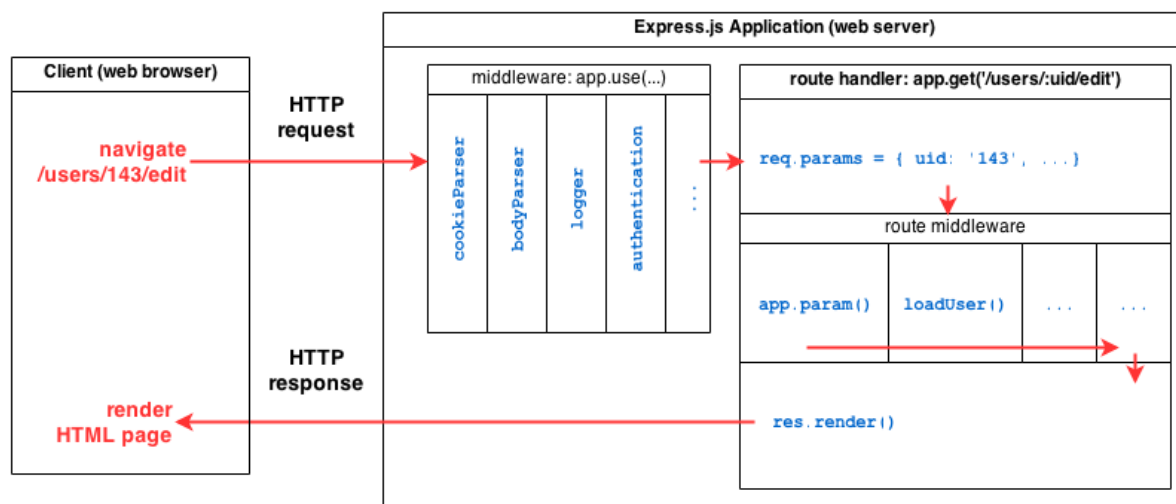


Рис. 1.10. Приклад проходження в Express запиту на зміну даних користувача

Клієнтський запит спочатку проходить кілька загальних `middleware`, обробників що виконуються на всіх запитах. Кожен з `middleware` виконує якісь свої окремі операції на кшталт зчитування `cookies`, логування, аутентифікація та передає цей запит через виклик функції `next()` наступному обробнику. Далі запит обробляється вже конкретно призначеним для нього роутом: парсяться

параметри запиту, виконується запит до БД, отримується відповідь від БД і вже фінальний результат операції надсилається назад клієнту.

1.4 Висновки до 1 розділу

У ході виконання кваліфікаційної роботи було успішно проведено дослідження, спроектовано та реалізовано інформаційну систему для обліку тварин у господарстві на базі сучасного MERN-стеку. Розроблений вебдодаток забезпечує автоматизацію процесу обліку поголів'я, що сприяє підвищенню точності та ефективності управлінських операцій, а також суттєво знижує ризики, пов'язані з людським фактором. В процесі роботи було детально проаналізовано сучасні технології веброботи, що дозволило обґрунтовано обрати стек MongoDB, Express.js, React.js та Node.js.

Створена система характеризується високою масштабованістю, зручністю для користувачів та готовністю до подальшого розширення функціоналу, зокрема шляхом інтеграції IoT-технологій, таких як RFID, сенсори та GPS. Це відкриває перспективи для більш глибокої автоматизації та моніторингу в реальному часі, що є надзвичайно важливим для сучасного аграрного сектору.

Таким чином, розроблений вебдодаток не лише сприяє цифровій трансформації тваринницької галузі, але й відповідає актуальним вимогам ринку щодо автоматизованих систем обліку, підвищуючи конкурентоспроможність господарств та оптимізуючи їхні операційні процеси.

Впровадження такої інформаційної системи відкриває нові можливості для аграрних підприємств у контексті розвитку «розумного» фермерства. Використання MERN-стеку забезпечує гнучкість і швидкість розробки, що дозволяє оперативно адаптувати систему під змінні потреби користувачів і технологічні інновації. Подальший розвиток проекту може включати впровадження аналітичних модулів на основі штучного інтелекту для прогнозування продуктивності, виявлення аномалій у стані тварин та

оптимізації ресурсів. Це зробить систему не лише інструментом обліку, а й потужним засобом підтримки прийняття управлінських рішень.

РОЗДІЛ 2

РОЗРОБКА МОДАЛЬНОГО ІНТЕРФЕЙСУ ВВЕДЕННЯ ОБ'ЄКТІВ У КЛІЄНТ-СЕРВЕРНОМУ ЗАСТОСУНКУ ДЛЯ АГРАРНОГО КОМПЛЕКСУ

Сучасний етап розвитку інформаційних технологій характеризується активним впровадженням вебзастосунків у різні сфери людської діяльності. Особливого значення це набуває у аграрному комплексі, де цифровізація процесів забезпечує підвищення ефективності управління, точності обліку ресурсів та оперативності прийняття рішень. Актуальність розробки інноваційних програмних засобів для аграрної галузі зумовлена необхідністю оптимізації збору, обробки та аналізу даних про об'єкти сільськогосподарського виробництва.

Одним із ключових аспектів побудови сучасних вебзастосунків є зручність та інтуїтивність інтерфейсу користувача. Застосування модальних вікон у інтерфейсі введення об'єктів дозволяє підвищити зручність взаємодії користувача із системою, зменшити кількість помилок під час введення даних та забезпечити логічну послідовність дій. Такі рішення сприяють не лише покращенню користувацького досвіду, а й підвищують продуктивність праці операторів системи.

Розробка клієнт-серверного вебзастосунку з модальним інтерфейсом введення об'єктів для аграрного комплексу забезпечує можливість централізованого зберігання даних, швидкого доступу до інформації та гнучкого керування нею в режимі реального часу. Це створює основу для побудови ефективних інформаційно-аналітичних систем у сільському господарстві, спрямованих на підтримку процесів планування, прогнозування та моніторингу діяльності.

2.1 Концепція та завдання розробки

Метою розробки є створення функціонального модуля збирання структурованої інформації через модальне вікно, здатного формувати об'єкт даних у форматі JSON і передавати його на сервер (бекенд) для подальшої обробки. Як приклад використано вебдодаток для ведення обліку тварин у господарстві, що спеціалізується на розведенні кролів.

Розроблюваний компонент реалізує введення даних нового елемента — «кролика» (*entity rabbit*), формуючи об'єкт із кількома ключовими властивостями: ім'я, порода, стать, мати, батько. Таке рішення забезпечує можливість легко масштабувати структуру об'єкта, доповнюючи її додатковими характеристиками (вік, дата народження, клітка тощо).

З технічного погляду в реалізації використано JavaScript як мову програмування, React.js — для створення модульного користувацького інтерфейсу, useState — для контролю стану полів, а також JSON як формат передачі структурованих даних у RESTful API. Архітектурна модель інтеграції зображена на рис. 2.1, де продемонстровано взаємодію трьох ключових частин системи: фронтенду, механізму формування об'єкта та бекенду.

2.2 Архітектура інтеграції модального інтерфейсу

На рис. 2.1 представлено структурну схему процесу введення об'єктів у клієнт-серверному застосунку.

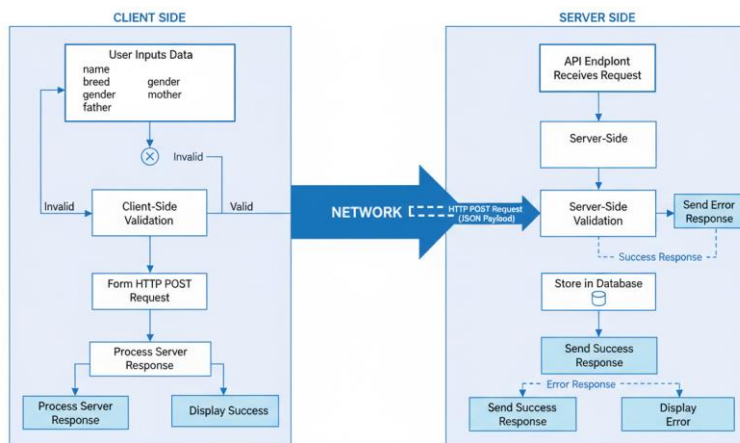


Рис. 2.1. Схема введення об'єктів у клієнт-серверному застосунку (авторська розробка)

Модальний інтерфейс у вебдодатку обліку тварин реалізовано як окремий клієнтський компонент, що інтегрується у загальну клієнт–серверну архітектуру застосунку на основі MERN-стеку. Його основним призначенням є введення, перевірка та передача даних про тварину на сервер з подальшою обробкою та збереженням у базі даних.

Згідно з представленою блок-схемою, архітектура інтеграції модального інтерфейсу поділяється на дві ключові частини: клієнтську та серверну.

Клієнтська частина

На клієнтському боці модальне вікно виконує функцію інтерфейсу введення даних користувачем. У формі модального вікна вводяться основні атрибути об'єкта (наприклад, ім'я, порода, стать, мати, батько). Після введення даних ініціюється етап клієнтської валідації, який забезпечує попередню перевірку коректності та повноти введеної інформації.

У разі виявлення помилок модальний інтерфейс блокує відправлення запиту та інформує користувача про некоректні поля. Якщо ж перевірка пройшла успішно, дані формуються у форматі JSON та передаються на сервер за допомогою HTTP POST-запиту через відповідний API-ендпоінт. Такий підхід зменшує навантаження на сервер та підвищує зручність взаємодії з користувачем.

Після отримання відповіді від сервера модальний інтерфейс обробляє результат: у разі успіху відображається повідомлення про успішне збереження даних або оновлюється інтерфейс, у разі помилки — виводиться відповідне повідомлення про помилку.

Серверна частина

На серверному боці архітектури інтеграції модального інтерфейсу реалізовано прийом та обробку запитів від клієнта через REST API. API-ендпоінт отримує HTTP POST-запит з JSON-навантаженням та передає дані на рівень серверної логіки.

Наступним етапом є серверна валідація, яка забезпечує перевірку цілісності, типів даних та бізнес-правил незалежно від клієнтської перевірки. У разі невідповідності даних сервер формує повідомлення про помилку та повертає його клієнту. Якщо ж валідація пройшла успішно, дані зберігаються у базі даних MongoDB.

Після завершення операції сервер формує відповідь про успішне виконання запиту та надсилає її клієнтській частині застосунку. Це дозволяє модальному інтерфейсу синхронізувати свій стан із сервером та забезпечити коректне відображення результату користувачу.

Узагальнення архітектурного підходу

Таким чином, інтеграція модального інтерфейсу в клієнт–серверну архітектуру вебдодатку базується на чіткому розмежуванні відповідальності між клієнтом і сервером, використанні двоступеневої валідації даних та асинхронній взаємодії через HTTP-запити. Запропонований підхід забезпечує масштабованість, надійність обробки даних і зручність користувацького інтерфейсу, що є важливим для подальшого розширення функціоналу системи.

2.3 Реалізація модального інтерфейсу введення даних

У реалізації передбачено використання декількох функцій-обробників подій, кожна з яких відповідає за оновлення конкретного поля форми. Наприклад:

1. `handleChangeBreed` — змінює породу;
2. `handleChangeGender` — визначає стать;

`handleChangeMother` і `handleChangeFather` — зчитують дані про батьків.

Ці функції дають змогу динамічно керувати станом, гарантуючи, що відображувані поля завжди утримують актуальні значення. Відповідно, після натискання кнопки “ADD” система формує цілісний об’єкт:

```
const rabbitToSend = {
```



```

    name,
    breed,
    gender,
    mother,
    father
  };

```

Закоментовані рядки (наприклад, `age`, `dateBirth`, `cage`) вказують на можливість розширення функціоналу без значних змін у логіці застосунку, що підкреслює масштабованість рішення. У момент відправлення натиснення кнопки «Save» викликає функцію, яка серіалізує об'єкт у формат JSON та виконує HTTP-запит до API-сервера:

```

fetch('/api/rabbits', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(rabbitToSend)
});

```

Таким чином відбувається асинхронна інтеграція клієнта з сервером, що є невід'ємним елементом сучасних SPA-застосунків (Single Page Application). Формування запиту не вимагає повного перезавантаження сторінки — це дозволяє забезпечити високу швидкодію та приємний користувацький досвід (UX).

2.4 Графічний вигляд і принципи UX-дизайну модального компонента

В роботі [1] розглянуто підхід до реалізації модального вікна для збирання даних на клієнтській частині вебзастосунку. Проаналізовано застосування функціональних компонентів на JavaScript у середовищі React для управління станом форми, а також побудову об'єкта, призначеного для серіалізації та надсилання на сервер. Роботу адаптовано до завдань розробки програмного забезпечення у форматі RESTful-клієнта з компонентним інтерфейсом. На рис. 2.2 зображено графічний інтерфейс модального вікна для додавання нового запису.

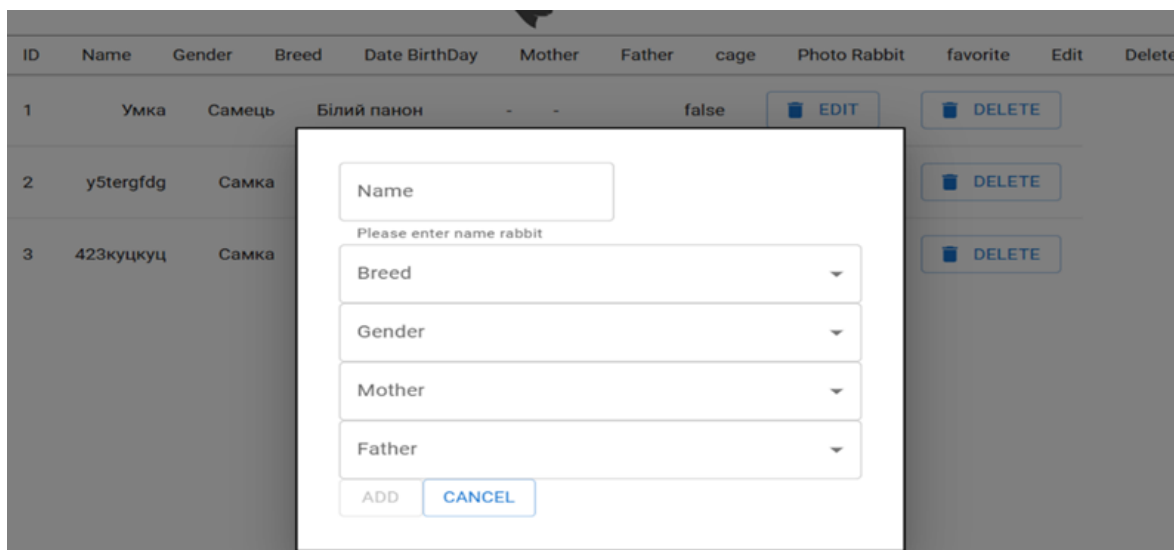


Рис. 2.2. Модальне вікно додавання нового запису — авторська розробка

Інтерфейс містить п'ять полів для введення інформації про тварину: Name, Breed, Gender, Mother, Father. Кожне поле реалізовано як контрольований елемент, який інтегровано з відповідною логікою через хуки React. Внизу форми знаходяться кнопки ADD (підтвердити внесення даних, яка неактивна до тих пір поки користувач не заповнить перших три важливих поля даними, а саме Name, Breed, Gender) та CANCEL (скасувати введення).

Модальне вікно побудоване з урахуванням принципів UX/UI-дизайну:

- мінімалізм і відсутність перевантажувальних елементів;
- чітка візуальна ієрархія полів;
- логічне розташування кнопок управління;
- адаптивність до різних розмірів екранів.

Завдяки цим підходам користувач може швидко вводити та коригувати інформацію без відволікання від основного контенту програми. Додатково реалізована валідація на клієнті (перевірка заповненості полів), щоб зменшити кількість помилок при відправці даних.

2.5 Переваги реалізованого рішення

Наведений підхід забезпечує низку суттєвих переваг для подальшого використання в агроінформаційних системах:

Масштабованість: структура об'єкта легко розширюється новими параметрами.

Прозорість обробки: завдяки контрольованим компонентам стан форми чітко синхронізується з логікою програми.

Модульність: компонент можна повторно використовувати в інших частинах системи або інтегрувати до інших вебзастосунків.

Продуктивність і асинхронність: використання HTTP-запитів без оновлення сторінки сприяє швидкодії навіть при великій кількості користувачів.

Зручність користування: зрозумілий інтерфейс зменшує когнітивне навантаження та прискорює роботу оператора.

2.6 Висновки до розділу 2

У результаті виконання розділу було розроблено модальний компонент введення даних, який інтегрується у клієнт-серверну архітектуру вебзастосунку для аграрної галузі. Продемонстровано механізм формування об'єкта даних у форматі JSON на прикладі структури «кролик». Детально розглянуто функціональну логіку React-компонента, принципи управління станом даних через хуки *useState* та передачу структурованої інформації на сервер через REST API.

Запропоноване рішення забезпечує гнучкість, розширюваність і сумісність із типовими архітектурними підходами вебпрограмування. Використання модальних вікон у системах аграрного обліку сприяє зручності роботи користувачів, підвищує швидкість взаємодії з базою даних і дозволяє реалізовувати сучасні інтерфейси з високим рівнем ергономічності. Такий підхід є оптимальним для систем, що потребують одночасно структурованості даних і зручного засобу їхнього введення.

РОЗДІЛ 3

ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБДОДАТКУ ОБЛІКУ ТВАРИН НА ОСНОВІ MERN-СТЕКУ

3.1. Загальна архітектура системи

Розроблений вебдодаток призначений для ведення обліку тварин у господарстві, ветеринарній клініці або іншій організації, що має потребу в систематизації інформації про тварин, їх медичний стан, історію вакцинацій та власників.

Основою реалізації застосунку є **MERN-стек** (MongoDB, Express.js, React, Node.js), що забезпечує повноцінний цикл розробки вебдодатків з використанням JavaScript як на клієнтській, так і на серверній сторонах.

Архітектура системи побудована за принципом клієнт-серверної взаємодії.

Frontend розроблений з використанням бібліотеки React, що дозволяє створити динамічний інтерфейс користувача з використанням компонентного підходу.

Backend реалізований на платформі Node.js із застосуванням фреймворку Express.js для побудови RESTful API.

База даних створена у MongoDB, що зберігає дані у неструктурованому форматі JSON-документів, що забезпечує гнучкість моделі даних.

3.2 Загальна структура Frontend

Клієнтська частина системи реалізована як односторінковий вебзастосунок (Single Page Application, SPA) з використанням бібліотеки React. Такий підхід дозволяє забезпечити швидку реакцію інтерфейсу, мінімізувати перезавантаження сторінок та покращити користувацький досвід.

Frontend-частина створена за допомогою інструменту Create React App, що надає стандартизовану структуру проєкту та налаштоване середовище розробки.

На рисунку 3.1 зображена типова структура клієнтського проєкту.

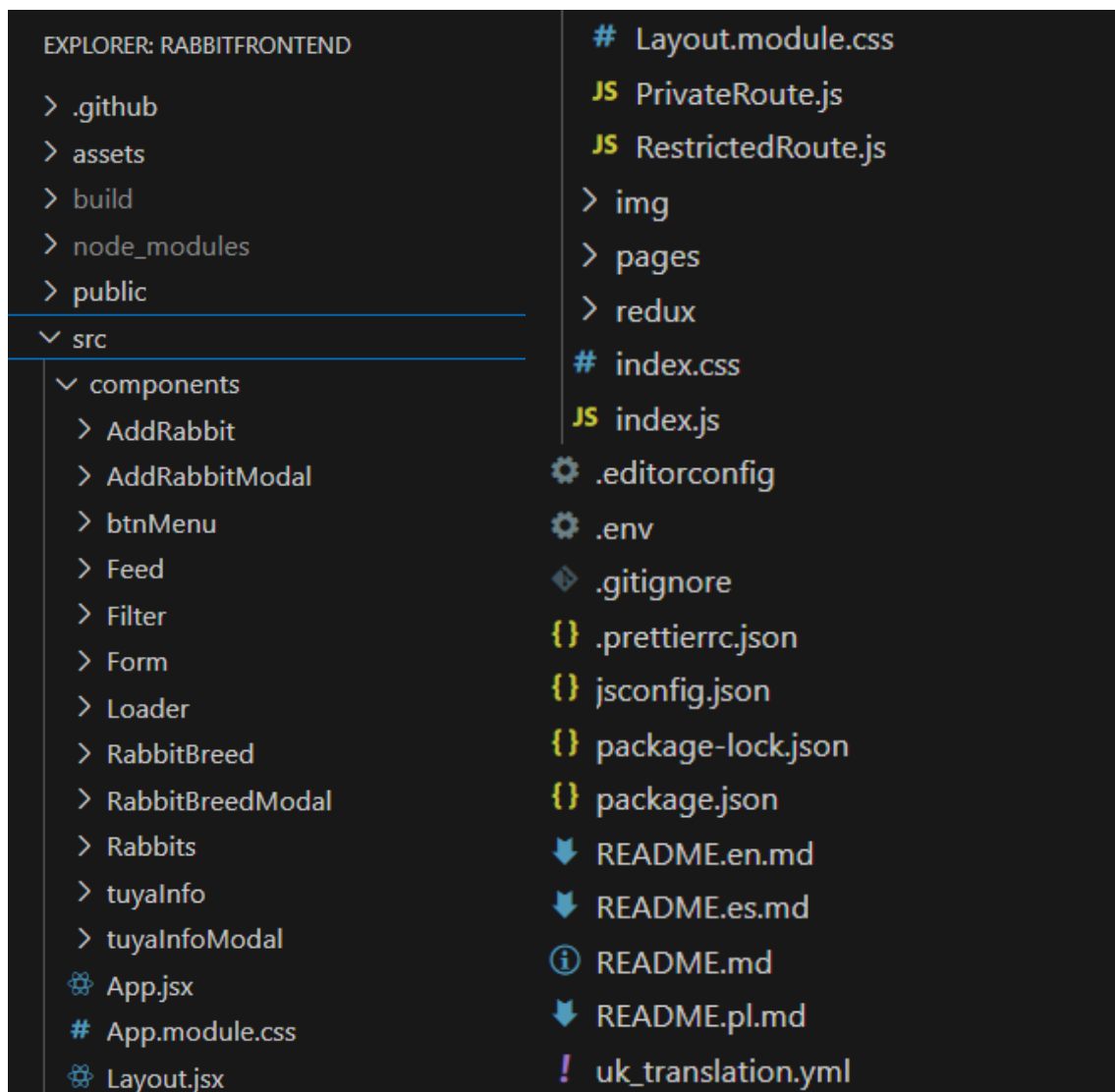


Рис. 3.1. Типова структура клієнтського проєкту.(Фронтенд)

3.2.1 Компонентна архітектура

Інтерфейс користувача побудований за компонентним підходом, де кожен компонент відповідає за окремий фрагмент інтерфейсу або логіки.

Компоненти (components):

- форми введення даних;
- таблиці для відображення записів ;

- кнопки, модальні вікна;
- елементи навігації.

Кожен компонент є незалежним і може повторно використовуватись у різних частинах застосунку.

3.2.2 Сторінки та маршрутизація

Для навігації між розділами застосунку використовується маршрутизація на клієнтській стороні. Кожна сторінка представляє окремий функціональний модуль системи, наприклад:

- перегляд списку тварин; [Додаток 1.]
- додавання нової тварини; [Додаток 2.]
- редагування існуючих записів;
- авторизація користувачів. [Додаток 3.]

Завдяки SPA-підходу перехід між сторінками відбувається без перезавантаження браузера.

3.2.3 Взаємодія з сервером

Frontend взаємодіє з Backend через окремий шар сервісів (services), який інкапсулює HTTP-запити до API. Це дозволяє:

- відокремити логіку інтерфейсу від роботи з сервером;
- спростити підтримку та тестування коду;
- централізовано обробляти помилки запитів.

Передача даних здійснюється у форматі JSON з використанням методів GET, POST, PUT, DELETE. [Додаток 4., Dodatok 5.]

3.3 Загальна структура Backend

Серверна частина системи відповідає за:

- обробку запитів клієнта;
- реалізацію бізнес-логіки;
- перевірку та валідацію даних;

- взаємодію з базою даних;
- забезпечення безпеки доступу до ресурсів.

Backend реалізований у вигляді **REST API**, що забезпечує універсальний інтерфейс взаємодії з клієнтською частиною.

На рисунку 3.1 зображена типова структура клієнтського проєкту.(Бекенд)

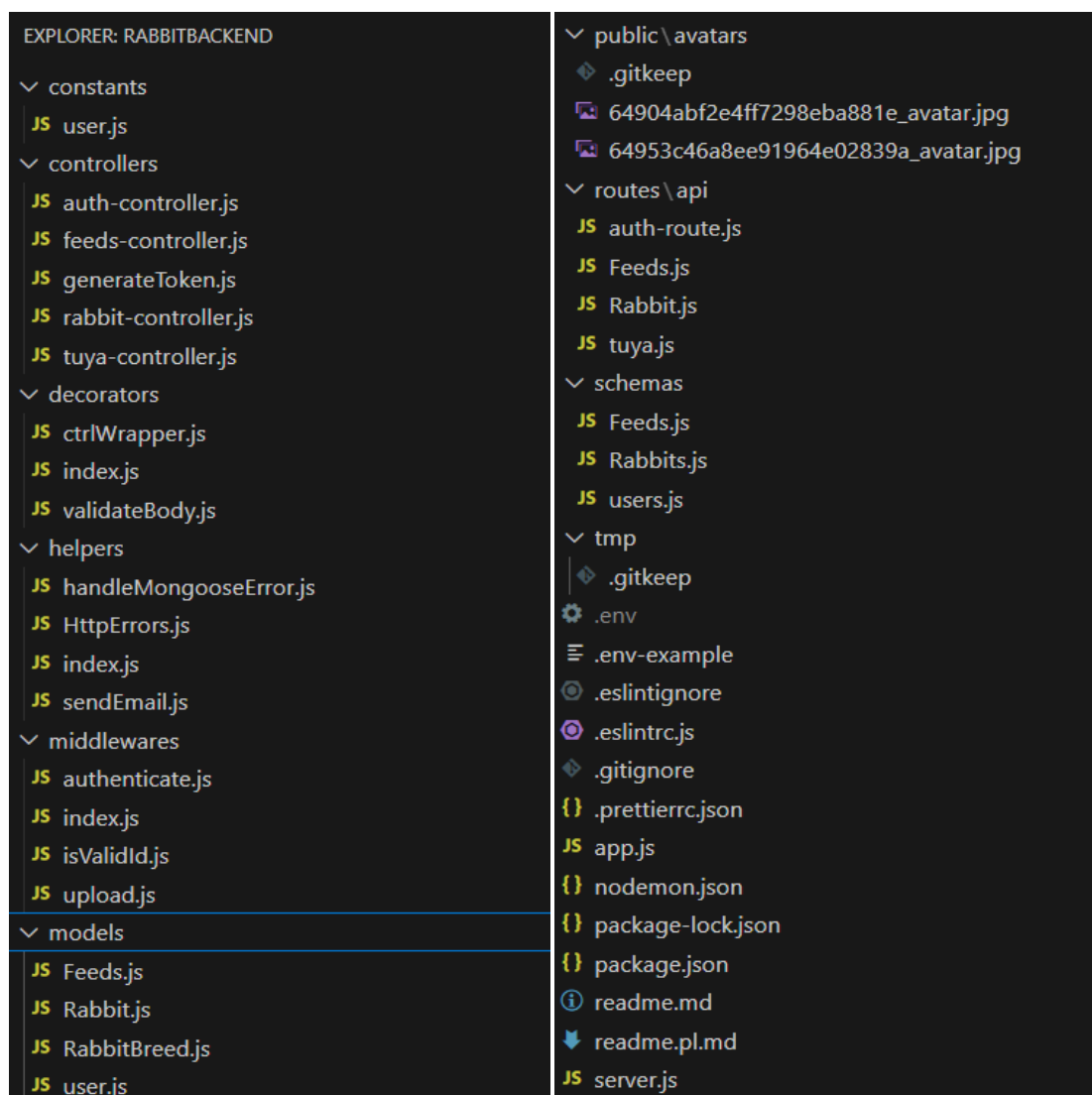


Рис. 3.1. Типова структура клієнтського проєкту. (Бекенд)

3.4 Реалізація та архітектурні особливості серверної частини вебдодатку

3.4.1. Призначення серверного рівня системи

Серверний рівень розробленого вебдодатку виконує функцію керування даними та забезпечує виконання основних операцій над інформаційними об'єктами системи. Бекенд відповідає за приймання запитів від клієнтського

інтерфейсу, їх логічну обробку та формування відповідей на основі актуального стану бази даних.

Застосування серверної обробки дозволяє винести критично важливу логіку за межі клієнтської частини, що позитивно впливає на безпеку, стабільність та керованість програмного продукту.

3.4.2. Технологічна основа бекенд-частини

Серверна частина реалізована на основі середовища виконання Node.js, яке забезпечує неблокуючу модель обробки вводу-виводу та ефективну роботу з мережевими запитами. Для організації серверної логіки застосовано фреймворк Express.js, який надає мінімалістичний, але гнучкий інструментарій для створення вебслужб.

Зберігання інформації організовано з використанням системи MongoDB, що дозволяє працювати з напівструктурованими даними. Для формалізації структури документів і взаємодії з базою даних використовується бібліотека Mongoose, яка забезпечує типізацію полів, опис зв'язків між об'єктами та перевірку коректності даних.

3.4.3. Організація серверного проєкту

Архітектура серверної частини побудована за принципом логічного розмежування компонентів. Кожен функціональний блок виконує окрему роль у загальному процесі обробки запитів.

У структурі бекенд-проєкту виділено такі рівні:

- модуль маршрутизації, що визначає точки доступу до ресурсів;
- рівень контролерів, який реалізує прикладну логіку;
- рівень моделей, що описує структуру збережених даних;
- допоміжні компоненти для попередньої обробки запитів та контролю помилок.

Зазначений підхід спрощує модифікацію окремих частин системи без впливу на її загальну працездатність.

3.4.4. Опис моделі обліку тварин

Центральним інформаційним об'єктом серверної частини є модель обліку тварин. Вона представляє собою сукупність атрибутів, необхідних для збереження та подальшого аналізу інформації про кожну тварину.

До складу моделі входять ідентифікаційні та описові характеристики, зокрема найменування, порода, статеві ознаки, а також дані про родинні зв'язки. Формалізація моделі за допомогою Mongoose-схеми дозволяє забезпечити цілісність даних та уніфікований підхід до їх обробки. [Додаток 6.]

3.4.5. Побудова програмного інтерфейсу взаємодії

Для забезпечення зв'язку між клієнтською та серверною частинами реалізовано програмний інтерфейс прикладного рівня, побудований за принципами REST. Взаємодія здійснюється шляхом використання стандартних HTTP-методів, що дозволяє виконувати операції створення, отримання, модифікації та видалення даних.

Кожен запит обробляється окремим серверним модулем, який здійснює аналіз вхідних параметрів, виконує необхідні дії з базою даних та формує відповідь у структурованому форматі. [Додаток 8.]

3.4.6. Контроль коректності та обробка помилок

У серверній частині передбачено механізми перевірки вхідної інформації, що надходить від клієнтської сторони. Це дозволяє виявляти помилки на ранніх етапах обробки та запобігати збереженню некоректних даних.

У разі виникнення нештатних ситуацій сервер формує відповідні повідомлення, які передаються клієнтській частині для інформування користувача або подальшої обробки. [Додаток 7.]

3.4.7. Особливості взаємодії з клієнтським інтерфейсом

Передача даних між сервером і клієнтським застосунком здійснюється у форматі JSON, що забезпечує незалежність сторін від внутрішньої реалізації одна одної. Серверна частина функціонує як окремий програмний сервіс, що дозволяє використовувати її спільно з різними клієнтськими рішеннями.

3.4.8. Узагальнення результатів реалізації бекенду

Реалізована серверна частина вебдодатку демонструє доцільність використання сучасних вебтехнологій для побудови інформаційних систем обліку. Запропонований підхід забезпечує структурованість, гнучкість та можливість подальшого розвитку системи без суттєвих архітектурних змін.

3.5. Висновки до розділу 3

У межах третього розділу виконано комплекс практичних робіт, спрямованих на створення та апробацію програмного рішення для автоматизованого ведення обліку тварин у господарстві. Реалізація поставлених завдань дала змогу перевірити застосовність сучасних вебтехнологій для розв'язання прикладних задач аграрної сфери та оцінити їх ефективність у реальних умовах використання.

У ході розробки сформовано багаторівневу структуру програмного продукту з чітким розмежуванням клієнтської та серверної логіки. Користувацький рівень побудовано з використанням компонентної моделі, що забезпечує логічну структурованість інтерфейсу, зручність навігації та можливість подальшого функціонального розширення без суттєвих змін архітектури. Обмін інформацією між частинами системи організовано за допомогою уніфікованих інтерфейсів взаємодії, що дозволило реалізувати асинхронну обробку запитів і підвищити швидкодію застосунку.

Серверний рівень реалізовано як окремий програмний модуль, відповідальний за обробку запитів, керування даними та підтримку цілісності інформації. Структура збереження даних спроектована з урахуванням специфіки предметної області, що забезпечило гнучкість модифікації схеми та спростило подальше супроводження системи. Реалізовані механізми перевірки даних і контролю помилок сприяють підвищенню стабільності роботи програмного забезпечення та зменшенню ризику некоректного введення інформації.

За результатами виконаної практичної частини отримано працездатний вебзастосунок, здатний забезпечити базові функції обліку та зберігання інформації про тварин. Запропоноване рішення може бути адаптоване до змінних умов експлуатації та слугувати основою для подальшого впровадження розширеного функціоналу, включно з інтеграцією зовнішніх сервісів і засобів автоматизованого збору даних.

Таким чином, результати розділу підтверджують можливість ефективного використання клієнт-серверних рішень для цифрової трансформації процесів у тваринництві та доцільність обраного підходу з погляду практичного застосування.

ВИСНОВКИ

У магістерській роботі здійснено комплексне дослідження можливостей використання сучасних вебтехнологій для цифровізації процесів обліку та управління у тваринницьких господарствах. Вибір тематики зумовлений актуальними тенденціями розвитку аграрного сектору, який дедалі активніше інтегрує інформаційні системи з метою підвищення ефективності виробництва, зниження операційних витрат і забезпечення прозорості управлінських рішень. Особливої значущості ця проблематика набуває в умовах необхідності оперативного опрацювання великих обсягів даних та мінімізації суб'єктивного впливу людського фактора.

У процесі виконання роботи було досягнуто поставленої мети — спроектовано та реалізовано клієнт-серверний вебзастосунок для обліку тварин у господарстві на основі MERN-стеку. Реалізоване програмне рішення підтвердило можливість створення повнофункціональної інформаційної системи з використанням єдиної технологічної екосистеми на базі мови JavaScript, що позитивно впливає на узгодженість компонентів, швидкість розробки та подальшу підтримку системи.

У першому розділі магістерської роботи проведено теоретичний аналіз розвитку вебтехнологій та підходів до створення вебзастосунків. Досліджено еволюцію архітектурних моделей, починаючи від статичних вебресурсів і завершуючи сучасними клієнт-серверними SPA-рішеннями. Окрему увагу приділено проблемам термінологічної невизначеності поняття «вебзастосунок», що дозволило узагальнити наявні наукові підходи та сформулювати власне бачення ролі вебдодатків у структурі сучасних інформаційних систем. Аналіз складових MERN-стеку засвідчив його відповідність вимогам масштабованості, гнучкості та продуктивності, що є критично важливим для аграрних інформаційних платформ.

Другий розділ роботи був спрямований на проєктування інтерфейсної частини системи, зокрема на розробку модального механізму введення даних у клієнт-серверному застосунку. У межах розділу сформульовано концептуальні принципи побудови інтерфейсу з урахуванням особливостей предметної області та потреб кінцевого користувача. Розроблений модальний компонент забезпечує структуроване введення інформації, зменшує ймовірність помилок та покращує загальну зручність роботи із системою. Реалізація інтерфейсу на основі сучасних UX-підходів сприяє підвищенню доступності функціоналу навіть для користувачів без спеціальної технічної підготовки.

У третьому розділі здійснено практичну реалізацію вебзастосунку та детально описано його архітектурні й технологічні особливості. Було спроектовано структуру клієнтської частини із застосуванням компонентного підходу, що дозволило розділити логіку відображення та обробки даних. Серверну частину реалізовано з використанням Node.js та Express.js, що забезпечило ефективну обробку HTTP-запитів і організацію REST API. Для збереження даних використано документоорієнтовану базу MongoDB, структура якої була адаптована до специфіки обліку тварин. Реалізовані механізми валідації даних і обробки помилок підвищують надійність та стабільність функціонування системи.

За результатами практичної частини створено працездатний програмний продукт, який дозволяє автоматизувати процеси збирання, збереження та перегляду інформації про тварин у господарстві. Розроблений вебзастосунок характеризується модульною архітектурою, що створює передумови для його подальшого розвитку та адаптації до змінних вимог предметної області. Отримані результати підтверджують доцільність використання клієнт-серверних рішень як інструменту цифрової трансформації аграрних процесів.

Практичне значення магістерської роботи полягає у можливості використання розробленого вебзастосунку в діяльності тваринницьких господарств різного масштабу для оптимізації облікових процесів та підвищення якості управлінських рішень. Окрім цього, матеріали роботи та реалізоване програмне рішення можуть бути використані в освітньому процесі під час вивчення сучасних вебтехнологій, архітектури інформаційних систем і принципів побудови клієнт-серверних застосунків.

Перспективи подальших досліджень пов'язані з розширенням функціональності системи шляхом впровадження аналітичних модулів, автоматизованих звітів і засобів прогнозування, а також інтеграції з IoT-пристроями для збору даних у реальному часі. Подальший розвиток такого програмного продукту може стати підґрунтям для створення комплексних інформаційних платформ аграрного призначення та сприяти поглибленню цифрової трансформації сільськогосподарської галузі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Карлов М.О., Козуб Г.О. Розробка модального інтерфейсу введення об'єктів у клієнт-серверному застосунку для аграрного комплексу. *Збірник наукових праць «ЛЮГОС» з матеріалами VII Міжнародної науково-практичної конференції, м. Болонья, 6 червня 2025 р.*—Вінниця-Болонья: ТОВ«УКРЛОГОС Груп», Associazione Italiana di Storia Urbana, 2025. С. 297-303 с . <https://doi.org/10.36074/logos-06.06.2025.057>.
2. Internet and social media users in the world 2024 | Statista. Statista. URL: <https://www.statista.com/statistics/617136/digital-population-worldwide>.
3. Newage. research 2023: the impact of war on users and media consumption trends. Newage. - Digital Advertising Agency. URL: <https://newage.agency/blog/newage-research-2023-the-impact-of-war-on-users-and-media-consumption-trends/>.
4. Haan K. Top website statistics for 2023. Forbes Advisor. URL: <https://www.forbes.com/advisor/business/software/website-statistics/>
5. Про авторське право і суміжні права : Закон України від 01.12.2022 р. № 2811-IX : станом на 15 квіт. 2023 р. URL: <https://zakon.rada.gov.ua/laws/show/2811-20#Text>
6. Український правопис / ред.: Є. І. Мазніченко та ін. Наук. думка, 2019. 390 с.
7. Про врахування висловлених органом державної реєстрації зауважень до наказу Міністерства цифрової трансформації України від 23 червня 2022 року № 57 : Наказ М-ва цифр. трансформації України від 21.07.2022 р. № 67. URL: <https://zakon.rada.gov.ua/laws/show/z0828-22#Text>
8. Словник української мови : у 20 т. / НАН України, Укр. мовно-інформ. фонд. — Київ : Наук. думка, 2010-. — (Словники України - Державна програма розвитку Національної словникової бази України). Т. 5 : 3 — Зв'язати / [уклад.: І. В. Шевченко та ін. ; наук. ред. О. О. Тараненко], 2014.

— 991 с. : табл. — Бібліогр. у підрядк. прим. — Покажч. словосполучень: с. 834–933.

9. Skrzypiec S., Plechawska-Wójcik M. Comparative analysis of Angular and React development frameworks. *Journal of computer sciences institute*. 2023. T. 28. C. 256–263. URL: <https://doi.org/10.35784/jcsi.3724>
10. Bielak K., Borek B., Plechawska-Wójcik M. Web application performance analysis using Angular, React and Vue.js frameworks. *Journal of computer sciences institute*. 2022. T. 23. C. 77–83. URL: <https://doi.org/10.35784/jcsi.2827>
11. Garcia V. H. Introduction to angular framework. *Getting started with angular*. Berkeley, CA, 2023. C. 1–11. URL: https://doi.org/10.1007/978-1-4842-9206-8_1
12. Progressive web app implementation in omah wayang klaten website / B. Susanto та ін. *Mobile computing and sustainable informatics*. Singapore, 2023. C. 333–348. URL: https://doi.org/10.1007/978-981-99-0835-6_24
13. Progressive web apps development and analysis with angular framework and service worker for e-commerce system / Z. Tahir та ін. *2021 IEEE international conference on computing (ICOCO)*, м. Kuala Lumpur, Malaysia, 17–19 листоп. 2021 р. 2021. URL: <https://doi.org/10.1109/icoco53166.2021.9673557>
14. Rojas C. Making your first progressive web app. *Building progressive web applications with vue.js*. Berkeley, CA, 2019. C. 1–46. URL: https://doi.org/10.1007/978-1-4842-5334-2_1
15. Baker M. *Secure web application development: a hands on guide with python and django*. Apress L. P., 2023.
16. Bartlett J. Creating a simple web app. *Building scalable PHP web applications using the cloud*. Berkeley, CA, 2019. C. 43–55. URL: https://doi.org/10.1007/978-1-4842-5212-3_4
17. Karunanidhi V. *Deploying tensorflow models to a web application*. Berkeley, CA : Apress, 2020. URL: <https://doi.org/10.1007/978-1-4842-6699-1>

18. Слабінога М. О., Чабан С. В. Розробка веб-додатків в контексті оптимізації їх швидкодії. Таврійський науковий вісник. Серія: технічні науки. 2022. № 3. С. 63–69. URL: <https://doi.org/10.32851/tnv-tech.2022.3.7>
19. Зосімов В. В. Моделі та засоби інтелектуальної обробки даних корпоративних веб-ресурсів : автореф. дис. д-ра техн. наук. Київ, 2019. 43 с.
20. Войтюк О. В., Плечистий Д. Д. Оптимізація промальовування вебзастосунку на основі об'єктів з глибокою вкладеністю та багатозалежними зв'язками. Технічна інженерія. 2023. № 1(91). С. 140–145. URL: [https://doi.org/10.26642/ten-2023-1\(91\)-140-145](https://doi.org/10.26642/ten-2023-1(91)-140-145)
21. Особливості розробки web-застосунків для системи дистанційного навчання з допомогою бібліотеки react / О. П. Кошова та ін. Systems and technologies. 2023. Т. 65, № 1. С. 20–31. URL: <https://doi.org/10.32782/2521-6643-2023.1-65.3>
22. Тлумачний словник з інформатики / Г.Г. Півняк, Б.С. Бусигін, М.М. Дівізінюк та ін. – Д., Нац. гірнич. ун-т, 2010. – 600 с.
23. Федько В. В., Безуглий Д. Є. Побудова веб застосунків на основі конструктивних елементів. Системи обробки інформації. 2020. № 1(160),. С. 123–127. URL: <https://doi.org/10.30748/soi.2020.160.16>
24. Базові поняття і терміни веб-технологій / [А. В. Кільченко, О. І. Поповський, О-р В. Тебенко, О-й. В. Тебенко, Н. М. Матросова]; Упорядник: Кільченко А. В. – К. : ІТЗН НАПН України, 2014. – 49 с.
25. Інформаційна технологія зберігання та візуалізації даних гідрометеорологічного прогнозування на основі WRF-Україна / С. Я. Майстренко, Т. О. Донцов-Загреба, К. В. Хурцилава, М. І. Харчук, С. В. Грибков, І. В. Ковалець // Математичні машини і системи. — Київ : ПІММС НАНУ, 2019. — № 1. — С. 56–67.
26. Матвеева Н., Нусс В. Використання фреймворку nuxt js 3 для розробки веб-додатку. Grail of science. 2023. № 25. С. 198–202. URL: <https://doi.org/10.36074/grail-of-science.17.03.2023.031>

27. Ужейко С. О. Способи відображення показників вітрової та сонячної енергії / С. О. Ужейко // Відновлювана енергетика. - 2017. - № 2. - С. 34-40. - Режим доступу: http://nbuv.gov.ua/UJRN/vien_2017_2_6.
28. Щербань В.Ю. Методи представлення, збереження та аналізу даних інформаційних систем / В.Ю. Щербань, С.М. Краснитський, Т.І. Астісова, В.М. Яхно. – К.: ТОВ "Фастбінд Україна", 2023. – 472 с.
29. Brooks D. W. Web-teaching: A guide for designing interactive teaching for the World Wide Web. 2-ге вид. New York : Kluwer Academic/Plenum Publishers, 2001. 331 с.
30. Baresi L., Garzotto F., Paolini P. From web sites to web applications: new issues for conceptual modeling. Conceptual Modeling for E-Business and the Web. Berlin, Heidelberg, 2000. С. 89–100. URL: https://doi.org/10.1007/3-540-45394-6_9
31. Belfrage A. Garden history and the web: dipping, dabbling, and diving. Australian garden history. 2011. 1 жовт. С. 13–16. URL: <https://www.jstor.org/stable/24918777>
32. What is a web app? - web application explained - AWS. Amazon Web Services, Inc. URL: <https://aws.amazon.com/what-is/web-application/>
33. Fowler S. L. Web application design handbook: best practices for web-based software. Amsterdam : Morgan Kaufmann Publishers, 2004. 658 с.
34. Hoffman A. Web application security: exploitation and countermeasures for modern web applications. O'Reilly Media, 2020. 330 с.
35. Brown E. Web development with node and express. O'Reilly Media, Incorporated, 2014.
36. Kappel G., Retschitzegger W., Schwinger W. Modeling customizable Web applications - a requirement's perspective. 2000 kyoto international conference on digital libraries: research and practice, м. Kyoto, Japan. URL: <https://doi.org/10.1109/dlrp.2000.942171>

37. Stack overflow. Stack Overflow Insights - Developer Hiring, Marketing, and User Research. URL: <https://insights.stackoverflow.com/trends?tags=iis,apache,express,nginx>.
38. Mejia A. Creating RESTful APIs with NodeJS and MongoDB Tutorial (Part II). Adrian Mejia Blog. URL: <https://adrianmejia.com/creating-a-restful-api-tutorial-with-nodejs-and-mongodb/>.
39. Java vs Nodejs: How to Choose the Right Technology. Belitsoft. URL: <https://belitsoft.com/java-development-services/java-vs-nodejs>.
40. Allazo, E.A.V., Cori, J.E.V. (2025). Agricultural Transformation: IoT Technology in the Controlled Cultivation of Oryza Sativa Seedlings in Greenhouse. *International Journal of Interactive Mobile Technologies (iJIM)*, 19(7), pp. 178–189. <https://doi.org/10.3991/ijim.v19i07.52851> Article submitted 2024-10-14
41. Java vs Nodejs: How to Choose the Right Technology. Belitsoft. <https://belitsoft.com/java-development-services/java-vs-nodejs>.
42. Mozilla Developer Network. JavaScript reference. (2025). URL: <https://developer.mozilla.org/>.
43. React. A JavaScript library for building user interfaces. <https://reactjs.org>
44. Rebenok, V., Rozhi, I., Petro, Y., Kozub, H., & Diachenko, N. (2024). Evolving information landscape: ICT's influence on societal digitalisation. *Multidisciplinary Science Journal*, 6, 2024ss0706. DOI: <https://doi.org/10.31893/multiscience.2024ss0706> . .
45. SmartBarn. Wireless Livestock Farm Alarm. <https://smartbarn.io/>
46. Wieruch, R. The Road to React. Open Source Edition, (2023). <https://github.com/VadymMakohon/eBook/blob/main/Wieruch%20R.%20-%20The%20Road%20to%20React%20-%202023.pdf> .
47. Козуб, Г. О., & Козуб, Ю. Г. (2022). Декларативний підхід при створенні мультиплатформних додатків. *Вісник Східноукраїнського національного*

- університету імені Володимира Даля, (5 (275)), 10–15.
<https://doi.org/10.33216/1998-7927-2022-275-5-10-15> .
48. Оленич, О. В., & Козуб, Г. О. (2025). Інтерфейс користувача як інструмент людино-машинної взаємодії: підходи та практика. «ΛΟΓΟΣ». Cambridge-Vinnytsia: P.C. Publishing House & UKRLOGOS Group LLC, 291-298.
<https://doi.org/10.36074/logos-09.05.2025.060>
49. Козуб Г.О., Козуб Ю.Г., Могильний Г.А., Жуков А.М. Розробка мобільного Android-додатку з застосуванням принципів Clean Architecture. *Вісник Східноукраїнського національного університету імені Володимира Даля*. вип. 5 (269), Вересень 2021, с. 5-10, doi:10.33216/1998-7927-2021-269-5-5-10.
50. Козуб Г., Попов Д. Особливості застосування вебтехнологій для систем керування освітніми закладами. *Ricerche scientifiche e metodi della loro realizzazione: esperienza mondiale e realtà domestiche: збірник наукових праць «ΛΟΓΟΣ» з матеріалами V Міжнародної науково-практичної конференції, м.Болонья, 2024р. Вінниця-Болонья: ТОВ«УКРЛОГОС Груп», Associazione Italianadi Storia Urbana, 2024. Pp.252-256. DOI: 10.36074/logos-26.04.2024.0252.*
51. Оленич О.В., Козуб Г.О. Інтерфейс користувача як інструмент людино-машинної взаємодії: підходи та практика. *Education and science of today: intersectoral issues and development of sciences: Collection of scientific papers «ΛΟΓΟΣ» with Proceedings of the VIII International Scientific and Practical Conference*, Cambridge, May9, 2025. Cambridge-Vinnytsia: P.C. Publishing House & UKRLOGOS Group LLC, 2025. pp. 291-298.
<https://doi.org/10.36074/logos-09.05.2025.060>
52. Вередін М., Козуб Г. Особливості UX/UI-дизайну при розробці вебсайтів та мобільних застосунків. *Матеріали конференцій МЦНД* (04 лип. 2025 р.; Ужгород, Україна). 2025. С. 122–125. <https://doi.org/10.62731/mcnd-04.07.2025.005>

53. Козуб Г.О., Сурма Ю. Ю., Артеменко О. І. Роль вебкомпонентів у побудові сучасних інтерфейсів: переваги та обмеження. *Вісник Херсонського національного технічного університету*. Том 2 № 2(93), 2025. С. 175-180. <https://doi.org/10.35546/kntu2078-4481.2025.2.2.21>
54. Statista. October 2025. URL: <https://www.statista.com/statistics/617136/digital-population-worldwide/>
55. Statista. From 2014 to October 2025. URL: <https://www.statista.com/statistics/325706/global-internet-user-penetration/>
56. Рівень проникнення інтернету в Україні сягнув 82,4% – дослідження Digital. URL: <https://fintechinsider.com.ua/riven-pronyknennya-internetu-v-ukrayini-syagnuv-824-doslidzhennya-digital>.

ДОДАТКИ

Додаток 1. Лістинг коду списку кроликів(Rabbits.jsx)

```
import { useSelector, useDispatch } from 'react-redux';
import RabbitsItem from '../Rabbits_item';
import { Loader } from '../Loader/Loader';
import { useEffect } from 'react';
import Table from '@mui/material/Table';
import TableCell from '@mui/material/TableCell';
import TableContainer from '@mui/material/TableContainer';
import TableBody from '@mui/material/TableBody';
import TableHead from '@mui/material/TableHead';
import TableRow from '@mui/material/TableRow';
import Paper from '@mui/material/Paper';

import {
  fetchRabbits,
  fetchRabbitsBreed,
} from '../../redux/rabbits/rabbitsOperation';
import {
  getRabbits,
  // getRabbitsBreed,
  // getError,
  getIsLoading,
} from '../../redux/rabbits/rabbitsSelector';

export const RabbitList = () => {
  const dispatch = useDispatch();

  // Отримуємо частини стану
  const { rabbits } = useSelector(getRabbits);
  const isLoading = useSelector(getIsLoading);
  // const error = useSelector(getError);

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

  return (
    <div>
      <div style={{ height: 400, width: '100%' }}>
        <Paper sx={{ width: '100%', overflow: 'hidden' }}>
          <TableContainer component={Paper}>
            <Table
              sx={{ minWidth: 650 }}
              size="small"
              aria-label="a dense table"
            >
```

```

>
<TableHead>
  <TableRow>
    <TableCell align="center">ID</TableCell>
    <TableCell align="right">Name</TableCell>
    <TableCell align="right">Gender</TableCell>
    <TableCell align="right">Breed</TableCell>
    <TableCell align="center">Date BirthDay</TableCell>
    <TableCell align="right">Mother</TableCell>
    <TableCell align="right">Father</TableCell>
    <TableCell align="right">cage</TableCell>
    <TableCell align="right">Photo Rabbit</TableCell>
    <TableCell align="right">favorite</TableCell>
    <TableCell align="center">Edit</TableCell>
    <TableCell align="center">Delete</TableCell>
  </TableRow>
</TableHead>
<TableBody>
  {isLoading ? (
    <TableRow>
      <TableCell colSpan={12} align="center">
        <Loader />
      </TableCell>
    </TableRow>
  ) : rabbits ? (
    <RabbitsItem Rabbits={rabbits} />
  ) : (
    <TableRow>
      <TableCell colSpan={12} align="center">
        Немає даних
      </TableCell>
    </TableRow>
  )}
</TableBody>
</Table>
</TableContainer>
</Paper>
</div>
</div>
);
};

```

Додаток 2. Лістинг коду модального вікна додавання кролика(AddRabbitModal.jsx)

```
import { useState, useEffect } from 'react';
import { useDispatch } from 'react-redux';
import * as MUI from '@mui/material/';
import { GetRabbitBreedList } from '../AddRabbit/GetRabbitBreedList';
import { RabbitGender } from '../AddRabbit/RabbitGender';
import { RabbitMother } from '../AddRabbit/RabbitMother';
import { RabbitFather } from '../AddRabbit/RabbitFather';
import { addRabbit, fetchRabbits } from '../../redux/rabbits/rabbitsOperation';

const style = {
  position: 'absolute',
  top: '50%',
  left: '50%',
  transform: 'translate(-50%, -50%)',
  width: 400,
  bgcolor: 'background.paper',
  border: '2px solid #000',
  boxShadow: 24,
  p: 4,
};

export default function AddRabbitModal({ openModal, closeModal }) {
  const [name, setName] = useState('');
  const [gender, setGender] = useState('');
  const [breed, setBreed] = useState('');
  const [mother, setMother] = useState('');
  const [father, setFather] = useState('');
  // const [cage, setCage] = useState('');
  // const [age, setAge] = useState('');
  // const [dateBirth, setDateBirth] = useState('');
  const [btn_check, setBtn_check] = useState(true);

  const dispatch = useDispatch();

  const rabbitToSend = {
    name,
    breed,
    gender,
    mother,
    father,
    // cage,
    // age,
    // dateBirth,
  };

  const handleChange = event => {
    switch (event.target.id) {
      case 'Name':
        setName(event.target.value);
        break;
      default:
        break;
    }
  };

  const handleChangeBreed = event => {
```



```

    setBreed(event.target.value);
  };

const handleChangeGender = event => {
  setGender(event.target.value);
};

const handleChangeMother = event => {
  setMother(event.target.value);
};

const handleChangeFather = event => {
  setFather(event.target.value);
};

useEffect(() => {
  if ((gender !== '') & (name !== '')) {
    setBtn_check(false);
  } else setBtn_check(true);
}, [btn_check, gender, name]);

const onSubmit = e => {
  e.preventDefault();
  dispatch(addRabbit(rabbitToSend))
    .then(e1 => (e1.payload === 201 ? closeModal() : ''))
    .then(() => dispatch(fetchRabbits()));
};

return (
  <div>
    <Mui.Modal
      open={openModal}
      onClose={closeModal}
      aria-labelledby="modal-modal-title"
      aria-describedby="modal-modal-description"
    >
      <Mui.Box sx={style}>
        <form onSubmit={onSubmit}>
          <div>
            <Mui.TextField
              helperText="Please enter name rabbit"
              id="Name"
              label="Name"
              onChange={handleChange}
            />
            {
              <GetRabbitBreedList
                breedSelect={breed}
                onChangeBreed={handleChangeBreed}
              />
            }
            {
              <RabbitGender
                gender={gender}
                onChangeGender={handleChangeGender}
              />
            }
            {
              <RabbitMother
                gender={mother}
                onChangeMother={handleChangeMother}

```

```

        />
    }
    {
        <RabbitFather
            gender={father}
            onChangeFather={handleChangeFather}
        />
    }
</div>
<div>
    <MUI.Button type="submit" disabled={btn_check} variant="outlined">
        Add
    </MUI.Button>
    <MUI.Button variant="outlined" onClick={closeModal}>
        Cancel
    </MUI.Button>
</div>
</form>
</MUI.Box>
</MUI.Modal>
</div>
);
}

```

Додаток 3. Лістинг коду авторизації користувача (Register.jsx)

```
import { useState } from 'react';
import { useDispatch } from 'react-redux';
import CSS from '../pages/Home.module.css';
import Button from '@mui/material/Button';
import TextField from '@mui/material/TextField';
import IconButton from '@mui/material/IconButton';
import OutlinedInput from '@mui/material/OutlinedInput';
import InputLabel from '@mui/material/InputLabel';
import InputAdornment from '@mui/material/InputAdornment';
import FormControl from '@mui/material/FormControl';
import Visibility from '@mui/icons-material/Visibility';
import VisibilityOff from '@mui/icons-material/VisibilityOff';
import { registerUser } from '../redux/auth/authOperation';

const Register = () => {
  const dispatch = useDispatch();
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [showPassword, setShowPassword] = useState(false);

  const handleClickShowPassword = () => setShowPassword(show => !show);

  const handleMouseDownPassword = event => {
    event.preventDefault();
  };

  const handleChange = ({ target: { name, value } }) => {
    switch (name) {
      case 'name': {
        setName(value);
        break;
      }
      case 'email': {
        setEmail(value);
        break;
      }
      case 'password': {
        setPassword(value);
        break;
      }
      default: {
        return value;
      }
    }
  };

  const regUser = e => {
    e.preventDefault();
    dispatch(registerUser({ name, email, password }));
    setName('');
    setEmail('');
    setPassword('');
  };

  return (
    <form onSubmit={regUser}>
      <div className={CSS.registration}>
```

```

<div className={CSS.items}>
  <h2>Регистрация</h2>
  <div className={CSS.items_input}>
    <TextField
      onChange={handleChange}
      id="demo-helper-text-name"
      label="Name"
      name="name"
      className={CSS.item_input}
    />
    <TextField
      onChange={handleChange}
      id="demo-helper-text-email"
      label="Email"
      name="email"
      className={CSS.item_input}
    />
    <FormControl variant="outlined" className={CSS.item_input}>
      <InputLabel htmlFor="outlined-adornment-
password">Password</InputLabel>
      <OutlinedInput
        id="outlined-adornment-password"
        name="password"
        type={showPassword ? 'text' : 'password'}
        onChange={handleChange}
        endAdornment={
          <InputAdornment position="end">
            <IconButton
              aria-label="toggle password visibility"
              onClick={handleClickShowPassword}
              onMouseDown={handleMouseDownPassword}
              edge="end"
            >
              {showPassword ? <VisibilityOff /> : <Visibility />}
            </IconButton>
          </InputAdornment>
        }
        label="Password"
      />
    </FormControl>
  </div>
  <Button variant="outlined" className={CSS.home_btn} type="submit">
    Регистрация
  </Button>
</div>
</div>
</form>
);
};

export default Register;

```

Додаток 4. Лістинг коду взаємодії з бекенд сервером

```
import axios from 'axios';
import { createAsyncThunk } from '@reduxjs/toolkit';
import Notiflix from 'notiflix';

// axios.defaults.baseURL = 'https://rabbitbackend.onrender.com';
axios.defaults.baseURL = 'http://localhost:3005';

const token = {
  set(token) {
    axios.defaults.headers.common.Authorization = `Bearer ${token}`;
  },
  unset() {
    axios.defaults.headers.common.Authorization = ``;
  },
};

export const fetchCurrentUser = createAsyncThunk(
  'auth/refresh',
  async (_, thunkAPI) => {
    const state = thunkAPI.getState();
    const persistToken = state.auth.token;

    if (persistToken === null) {
      // console.log('Токена не існує');
      return thunkAPI.rejectWithValue();
    }

    token.set(persistToken);
    try {
      const { data, status } = await axios.get('/users/current');
      if (status === 401) token.unset();
      return data;
    } catch (err) {
      return thunkAPI.rejectWithValue(err);
    }
  }
);

export const registerUser = createAsyncThunk(
  'auth/registration',
  async (user, { rejectWithValue }) => {
    try {
      const { data, status } = await axios.post('/users/register', user);
      if (status === 201)
        Notiflix.Notify.success('Юзер успішно зареєстрований');
      token.set(data.token);
      return data;
    } catch (err) {
      Notiflix.Notify.failure('Щось пішло не так!!! Помилка: ' + err.message);
      return rejectWithValue(err.message);
    }
  }
);

export const logInUser = createAsyncThunk(
  'users/login',
  async (user, { rejectWithValue }) => {
    try {
```

```

    const { data } = await axios.post('/users/login', user);
    token.set(data.token);
    return data;
  } catch (err) {
    if (err.response.status === 401) {
      Notiflix.Notify.failure('невірний логін або пароль');
    } else
      Notiflix.Notify.failure(
        'Щось пішло не так!!! Помилка: ' + err.response.status
      );
    return rejectWithValue(err.response.status);
    // return err;
  }
}
);

export const logOutUser = createAsyncThunk('auth/logout', async () => {
  try {
    const { status } = await axios.post('/users/logout');
    if (status === 200) Notiflix.Notify.success('Ви розлогінілись');
    token.unset();
  } catch (err) {
    Notiflix.Notify.failure('Щось пішло не так!!! Помилка: ' + err.message);
    return err.message;
  }
});

```

Додаток 5. Лістинг коду взаємодії з бекенд сервером(feedOperation.js)

```
import axios from 'axios';
import { createAsyncThunk } from '@reduxjs/toolkit';
import Notiflix from 'notiflix';

// axios.defaults.baseURL = 'https://rabbitbackend.onrender.com';
axios.defaults.baseURL = 'http://localhost:3005';

const token = {
  set(token) {
    axios.defaults.headers.common.Authorization = `Bearer ${token}`;
  },
  unset() {
    axios.defaults.headers.common.Authorization = ``;
  },
};

//////////Feeds SET//////////
export const addFeed = createAsyncThunk(
  'feeds/addFeeds',
  async (feed, thunkAPI) => {
    const state = thunkAPI.getState();
    const persistToken = state.auth.token;
    token.set(persistToken);

    try {
      const { status } = await axios.post('/feeds/add', feed);
      if (status === 201)
        Notiflix.Notify.success(
          `Корм додано у базу! ${'\n'} The feed was successfully created.`
        );
      return status;
    } catch (err) {
      Notiflix.Notify.failure(err.message);
      return err.message;
    }
  }
);

export const fetchFeeds = createAsyncThunk('feeds/refresh', async (_, thunkAPI)
=> {
  const state = thunkAPI.getState();
  const persistToken = state.auth.token;

  if (persistToken === null) {
    // console.log('Токена не існує');
    return thunkAPI.rejectWithValue();
  }

  token.set(persistToken);
  try {
    const first = await axios.get('feeds');
    if (first.status === 401) token.unset();
    const data = [first.data];
    return data;
  } catch (err) {
    return thunkAPI.rejectWithValue(err);
  }
});
```

```

export const findCurrentFeedById = createAsyncThunk(
  'feeds/findCurrentFeedById',
  async (id, { rejectWithValue }) => {
    try {
      const { data, status } = await axios.get(`/rabbits/breeds/${id}`);
      if (status === 200) return data;
    } catch (err) {
      Notiflix.Notify.failure(err.message);
      return rejectWithValue(err.message);
    }
  }
);

```

```

export const updateFeed = createAsyncThunk(
  'feeds/updateFeed',
  async (objectsToSend, thunkAPI) => {
    console.log('object', objectsToSend[1]);
    console.log('id', objectsToSend[0]);
    const state = thunkAPI.getState();
    const persistToken = state.auth.token;
    token.set(persistToken);

    try {
      const { data, status } = await axios.put(
        `/rabbits/breeds/${objectsToSend[0]}`,
        objectsToSend[1]
      );
      if (status === 200)
        Notiflix.Notify.success(
          'Корм редаговано у бази. \n The feed was successfully updated.'
        );
      return data;
    } catch (err) {
      Notiflix.Notify.failure(err.message);
      // return rejectWithValue(err.message);
    }
  }
);

```

```

export const deleteFeed = createAsyncThunk(
  'feeds/deleteFeed',
  async (id, { rejectWithValue }) => {
    try {
      const { status } = await axios.delete(`/feeds/delete/${id}`);
      if (status === 200)
        Notiflix.Notify.success(
          `Корм видалено з бази! \n The feed was successfully deleted.`
        );
      return status;
    } catch (err) {
      Notiflix.Notify.failure(err.message);
      return rejectWithValue(err.message);
    }
  }
);

```


Додаток 6. Лістинг коду бекенду Mongoose-схеми (Rabbits.js)

```

const Joi = require('joi');

const rabbitSchema = Joi.object()
  .min(1)
  .keys({
    name: Joi.string().min(2).required().messages({
      'any.required': `missing required 'name' field`,
      'string.empty': ` 'name' cannot be an empty field`,
    }),

    breed: Joi.string().min(6).required().messages({
      'any.required': `missing required 'breed' field`,
      'string.empty': ` 'breed' cannot be an empty field`,
    }),

    gender: Joi.string().min(4).required().messages({
      'any.required': `missing required 'gender' field`,
      'string.empty': ` 'gender' cannot be an empty field`,
    }),

    mother: Joi.string().messages({
      'any.required': `missing required 'mother' field`,
      'string.empty': ` 'mother' cannot be an empty field`,
    }),

    father: Joi.string().messages({
      'any.required': `missing required 'father' field`,
      'string.empty': ` 'father' cannot be an empty field`,
    }),

    // dateBirthDay: Joi.string().min(6).required().messages({
    //   'any.required': `missing required 'dateBirthDay' field`,
    //   'string.empty': ` 'dateBirthDay' cannot be an empty field`,
    // }),

    // favorite: Joi.boolean(),
  });

const rabbitBreedSchema = Joi.object()
  .min(1)
  .keys({
    name: Joi.string().min(2).required().messages({
      'any.required': `missing required 'name' field`,
      'string.empty': ` 'name' cannot be an empty field`,
    }),

    color: Joi.string().min(4).required().messages({
      'any.required': `missing required 'color' field`,
      'string.empty': ` 'color' cannot be an empty field`,
    }),

    about: Joi.string().messages({
      'string.empty': ` 'about' cannot be an empty field`,
      'any.required': `missing required 'about' field`,
    }),

    // favorite: Joi.boolean(),
  });

module.exports = { rabbitSchema, rabbitBreedSchema };

```

Додаток 7. Лістинг коду бекенду модель-схеми (../models/Rabbits.js)

```

const { Schema, model } = require('mongoose');

const { handleMongooseError } = require('../helpers');

const RabbitSchema = new Schema(
  {
    name: {
      type: String,
      required: [true, 'Set name for rabbit'],
    },

    gender: {
      type: String,
      required: [true, 'Set gender for rabbit'],
    },

    breed: {
      type: String,
      required: [true, 'Set breed for rabbit'],
    },

    // dateBirthDay: {
    //   type: Date,
    //   required: [true, 'Set date of BirthDay for rabbit'],
    //   default: '00/00/2023',
    // },

    mother: {
      type: String,
      // required: [false, 'Set Mother for rabbit'],
      default: 'unknown',
    },

    father: {
      type: Object,
      // required: [false, 'Set Father for rabbit'],
      default: 'unknown',
    },

    Vakcine: {
      type: Array,
    },

    weight: {
      type: Array,
    },

    // cage: {
    //   type: Array,
    //   required: [false, 'Set cage for rabbit'],
    // },

    favorite: {
      type: Boolean,
      default: false,
    },

    photoRabbit: {
      type: String,
      default: '',
    }
  }
);

```

```
    },  
    owner: {  
      type: Schema.Types.ObjectId,  
      ref: 'user',  
      required: true,  
    },  
  },  
  { versionKey: false, timestamps: true }  
);  
  
RabbitSchema.post('save', handleMongooseError);  
  
const Rabbit = model('Rabbit', RabbitSchema);  
  
module.exports = Rabbit;
```

Додаток 8. Лістинг коду бекенду контролера (../controllers/rabbits-controller.js)

```

const Rabbit = require('../models/Rabbit');
const RabbitsBreed = require('../models/RabbitBreed');
const { HttpError } = require('../helpers');
const { ctrlWrapper } = require('../decorators');
// const { query } = require('express');

const getAllRabbits = async (req, res) => {
  const { _id: owner } = req.user;
  const { ...query } = req.query;
  const resultList = await Rabbit.find({ owner, ...query }, '-createdAt -
updatedAt').populate('name', 'breed');
  res.json(resultList);
};

const getRabbitById = async (req, res) => {
  const rabbitId = req.params.rabbitId;
  const getRabbitResult = await Rabbit.findById(rabbitId);
  if (!getRabbitResult) {
    throw HttpError(404);
  }
  res.json(getRabbitResult);
};

const addRabbit = async (req, res) => {
  const { _id: owner } = req.user;
  const addRabbitResult = await Rabbit.create({ ...req.body, owner });
  res.status(201).json(addRabbitResult);
};

const updateRabbit = async (req, res) => {
  const rabbitId = req.params.rabbitId;
  const updateRabbitResult = await Rabbit.findByIdAndUpdate(rabbitId, req.body,
{ new: true });
  if (!updateRabbitResult) {
    throw HttpError(404);
  } else res.json(updateRabbitResult);
};

const updateRabbitFavorite = async (req, res) => {
  const rabbitId = req.params.rabbitId;
  const updateRabbitResult = await Rabbit.findByIdAndUpdate(rabbitId, req.body,
{ new: true });
  if (!updateRabbitResult) {
    throw HttpError(404);
  } else res.json(updateRabbitResult);
};

const deleteRabbit = async (req, res) => {
  const RabbitId = req.params.rabbitId;
  const deleteRabbitResult = await Rabbit.findByIdAndDelete(RabbitId);
  if (deleteRabbitResult === null) {
    throw HttpError(404);
  } else res.status(200).json({ message: 'Rabbit deleted' });
};

```

```
//
////////////////////////////////////BREED////////////////////////////////////
const getAllRabbitsBreed = async (req, res) => {
  const { _id: owner } = req.user;
  const { ...query } = req.query;
  const resultList = await RabbitsBreed.find({ owner, ...query }, '-createdAt -
updatedAt').populate('name', 'color');
  // console.log('resultBreed>>>>>', resultList);
  res.json(resultList);
};

const addRabbitBreed = async (req, res) => {
  const { _id: owner } = req.user;
  const addRabbitResult = await RabbitsBreed.create({ ...req.body, owner });
  res.status(201).json(addRabbitResult);
  return res.status(201);
};

const getRabbitBreedById = async (req, res) => {
  const rabbitBreedId = req.params.breedId;
  const getRabbitResult = await RabbitsBreed.findById(rabbitBreedId);
  if (!getRabbitResult) {
    throw HttpError(404);
  }
  res.json(getRabbitResult);
};

const updateRabbitBreed = async (req, res) => {
  const rabbitBreedId = req.params.breedId;
  const updateRabbitResult = await RabbitsBreed.findByIdAndUpdate(rabbitBreedId,
req.body, { new: true });
  if (!updateRabbitResult) {
    throw HttpError(404);
  } else res.json(updateRabbitResult);
};

const deleteRabbitBreed = async (req, res) => {
  const BreedId = req.params.breedId;
  const deleteResult = await RabbitsBreed.findByIdAndDelete(BreedId);
  if (deleteResult === null) {
    throw HttpError(404);
  } else res.status(200).json({ message: 'Breed deleted' });
};

module.exports = {
  getAllRabbits: ctrlWrapper(getAllRabbits),
  getAllRabbitsBreed: ctrlWrapper(getAllRabbitsBreed),
  getRabbitById: ctrlWrapper(getRabbitById),
  getRabbitBreedById: ctrlWrapper(getRabbitBreedById),
  addRabbit: ctrlWrapper(addRabbit),
  addRabbitBreed: ctrlWrapper(addRabbitBreed),
  deleteRabbit: ctrlWrapper(deleteRabbit),
  deleteRabbitBreed: ctrlWrapper(deleteRabbitBreed),
  updateRabbit: ctrlWrapper(updateRabbit),
  updateRabbitBreed: ctrlWrapper(updateRabbitBreed),
  updateRabbitFavorite: ctrlWrapper(updateRabbitFavorite),
};
```

CERTIFICATO DI PARTECIPAZIONE

Maxim Karlov

ha partecipato alla VII Conferenza scientifica e pratica internazionale

**«Ricerche scientifiche e metodi della loro realizzazione:
esperienza mondiale e realtà domestiche»**

e pubblicato articoli scientifici

**РОЗРОБКА МОДАЛЬНОГО
ІНТЕРФЕЙСУ ВВЕДЕННЯ
ОБ'ЄКТІВ У
КЛІЄНТ-СЕРВЕРНОМУ
ЗАСТОСУНКУ ДЛЯ АГРАРНОГО
КОМПЛЕКСУ**

DAVIDE SQUARCIACIA
Manager delle Pubblicazioni
In Viaggio Con il Levi

MIRIAM GOLDENBLAT
Capo della Piattaforma
Presidente del comitato

- ✓ La conferenza è inclusa nell'**ACADEMIC RESEARCH INDEX**
- ✓ La conferenza è certificata secondo lo standard SCC-2000
- Euro Science Certificato**
N° 22903 del 03.04.2025
- ✓ La conferenza è registrata presso l'Istituto ucraino di competenza e informazione scientifica e tecnica». **UKRISTEI** Certificato N° 418 del 12.06.2024
- ✓ libretto rilasciato secondo le norme ISO 2108:2005, ISO 10863:1991 e ISO 7275:1985.
- ✓ articolo pubblicato indicizzato in CrossRef, Google Scholar, OUCI, OpenAIRE, WorldCat, Semantic Scholar, Mendeley, Scilit, PubPeer, Lens.org, Scite, ecc.

ISPC N° 060625-030

Bologna
Repubblica Italiana

6 giugno
2025

0.6 ECTS credits (18 hours)
Recommended by the Academic Council of the «Institute of Scientific and Technical Integration and Cooperation»
Protocol N° 22 from June 5th, 2025.

Atti della Conferenza scientifica e pratica internazionale sono pubblicati nella Raccolta di articoli scientifici ADTOE.

DOI 10.36074/10gos-06.06.2025