МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ВЕБЗАСТОСУНКУ ЗАБЕЗПЕЧЕННЯ БІЗНЕСУ З ВИКОРИСТАННЯМ REACT ТА NODE.JS»**

Виконав: студент 4 курсу, групи 6.1211-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
О.Ю. Лукашев
(ініціали та прізвище)
Керівник доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н. Панасенко Є.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Лукашеву Олесю Юрійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебзастосунку забезпечення бізнесу з використанням React та Node.js

керівник роботи Панасенко Євген Валерійович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Аналіз вимог та розробка вебсайту архітектурної майстерні.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 26.12.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.01.2025	
2.	Збір вихідних даних.	07.02.2025	
3.	Обробка методичних та теоретичних	17.03.2025	
	джерел.		
4.	Розробка першого та другого розділу.	21.04.2025	
5.	Розробка третього розділу.	19.05.2025	
6.	Оформлення та нормоконтроль	26.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	17.06.2025	

Студент _____ О.Ю. Лукашев
(підпис) (ініціали та прізвище)

Керівник роботи _____ Є.В. Панасенко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка вебзастосунку забезпечення бізнесу з використанням React та Node.js»: 56 с., 31 рис., 1 табл., 20 джерел.

АРХІТЕКТУРНА МАЙСТЕРНЯ, БАЗА ДАНИХ, ІНТЕРФЕЙС КОРИСТУВАЧА, КОНТАКТНА ФОРМА, NODE.JS, REACT-КОМПОНЕНТИ.

Об’єктом дослідження є процес створення сучасних вебзастосунків для представлення діяльності творчих студій. Предметом – технології розробки застосунку-візитівки з використанням React для клієнтської частини та Node.js для серверної.

Мета роботи полягає у створенні адаптивного та функціонального вебзастосунку для архітектурної майстерні “Lukashev architects” із можливістю залишати контактні дані для зворотного зв’язку.

Методи дослідження: аналіз предметної області, порівняння технологій, реалізація застосунку на основі отриманих результатів.

У рамках роботи створено повноцінний вебресурс із динамічним інтерфейсом, серверною логікою та базою даних PostgreSQL для збереження звернень користувачів. Визначено ефективні інструменти веброзробки, які сприяють покращенню комунікації з клієнтами, автоматизації обробки запитів та презентації бренду архітектурної студії.

Робота пов’язана з дослідженнями у сфері створення вебрішень для малого бізнесу й творчих команд, із врахуванням принципів UX/UI, брендингу та адаптивного дизайну.

SUMMARY

Bachelor’s qualifying paper “Development of a Business Support Web Application Using the React and Node.js”: 56 pages, 31 figures, 1 table, 20 references.

ARCHITECTURAL STUDIO, CONTACT FORM, DATABASE, NODE.JS, REACT COMPONENTS, USER INTERFACE.

The object of this study is the process of creating modern web applications for creative businesses, particularly architectural studios. The subject is the use of React for the frontend and Node.js for the backend in developing a portfolio web application.

The aim is to create a responsive, user-friendly web application for “Lukashev Architects”, featuring a contact form for feedback and client interaction.

The research methods include domain analysis, comparison of modern technologies, and practical implementation based on findings.

The developed solution uses React for a dynamic interface and Node.js for server logic, with PostgreSQL for storing contact requests. The work analyzes current tools that improve brand presentation, client communication, and the automation of data collection.

This paper builds on prior studies related to web development for small creative businesses, with attention to branding, UX/UI design, and functionality.

The application is focused on showcasing projects, simplifying navigation, and ensuring adaptability. Given the importance of online presence for design studios, this solution supports branding, communication, and outreach.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 7
1 Аналіз вимог та планування проєкту 9
1.1 Аналіз вимог замовника 9
1.2 Формулювання вимог до системи 11
1.3 Вибір клієнтських технологій та побудова інтерфейсу 13
1.4 Серверне середовище Node.js та каркас Express.js у веброзробці 16
1.5 Реляційна база даних PostgreSQL та її переваги..... 18
1.6 Порівняльна характеристика альтернативних технологій..... 20
2 Проєктування вебзастосунку 24
2.1 Проєктування архітектури вебзастосунку..... 24
2.2 Проєктування інтерфейсу користувача 26
2.3 Налаштування серверної частини вебзастосунку..... 30
2.4 Створення ER-діаграми бази даних 33
3 Реалізація та тестування вебзастосунка..... 36
3.1 Розробка фронтенду з використанням React.js 36
3.2 Розробка бекенду з використанням Node.js 43
3.3 Тестування вебзастосунку..... 46
3.4 Результати реалізації та тестування 51
Висновки 54
Перелік посилань..... 55

ВСТУП

Сьогодні інформаційні технології відіграють ключову роль у розвитку бізнесу, особливо в креативних галузях, таких як архітектура та дизайн. У світі, де цифрова присутність є необхідною умовою конкурентоспроможності, ефективна онлайн-комунікація з клієнтами, партнерами та громадськістю суттєво впливає на успішність компанії. Вебзастосунки дозволяють не лише представляти послуги, а й формувати імідж, демонструвати портфоліо, передавати цінності бренду та забезпечувати швидкий зворотний зв'язок.

Якщо раніше ознайомлення з роботою архітектурної студії передбачало особисті зустрічі чи перегляд друкованих матеріалів, то нині сучасний вебзастосунок дає змогу потенційному замовнику легко ознайомитися з реалізованими проєктами, філософією студії, принципами діяльності та контактною інформацією в зручному онлайн-форматі. Тому створення якісного, функціонального і привабливого вебзастосунку-візитівки для архітектурної студії є актуальним і практично важливим завданням.

Метою цієї кваліфікаційної роботи є розробка вебзастосунку-візитівки для архітектурної майстерні "Lukashev Architects" із використанням сучасних вебтехнологій: бібліотеки React для клієнтської частини та середовища Node.js для серверної логіки.

Для досягнення мети необхідно виконати такі завдання:

- дослідити вимоги до цифрової презентації архітектурних послуг;
- проаналізувати можливості React і Node.js для створення SPA-застосунків;
- спроєктувати архітектуру застосунку, структуру, навігацію та дизайн;
- реалізувати клієнтську частину з інтерактивними елементами, портфоліо та контактною формою;
- налаштувати серверну частину, зберігання повідомлень у базі даних;

– провести тестування, перевірити стабільність, продуктивність і відповідність вимогам.

Об’єкт дослідження – процес створення вебзастосунків для презентації діяльності у сфері архітектури. Предмет – технології реалізації вебвізитівок із використанням React та Node.js.

Методи дослідження включають аналіз літератури та документації, огляд сучасних рішень, порівняння технологій веброзробки та практичне впровадження.

Для реалізації застосунку обрано JavaScript-стек: React – для створення інтерфейсу користувача, Node.js – для серверної логіки. Це забезпечує узгоджену взаємодію між частинами, спрощує розробку та супровід. Дані з контактної форми зберігаються в PostgreSQL – надійній СУБД із підтримкою реляційної структури. Взаємодія клієнта і сервера відбувається через REST API, що дозволяє масштабувати систему.

Кваліфікаційна робота складається з трьох розділів. У вступі обґрунтовано актуальність теми, сформульовано мету, завдання, об’єкт і предмет дослідження. Перший розділ присвячено теоретичним основам веброзробки, опису React і Node.js, а також огляду альтернативних технологій. У другому розділі розглянуто проєктування архітектури застосунку, структуру бази даних і логіку обробки запитів. Третій розділ містить опис практичної реалізації, тестування, аналіз результатів і оцінку ефективності розробленого рішення.

1 АНАЛІЗ ВИМОГ ТА ПЛАНУВАННЯ ПРОЄКТУ

1.1 Аналіз вимог замовника

Замовником розробки вебзастосунку є архітектурна студія “Lukashev Architects”, яка прагне посилити свою цифрову присутність і вдосконалити комунікацію з потенційними клієнтами через створення сучасного, естетично вивіреного та технічно досконалого вебсайту. Під час початкового етапу реалізації проєкту було проведено глибоке обговорення потреб студії, визначено очікування щодо функціоналу, дизайну, структури ресурсу та можливостей його подальшого розвитку. У результаті спілкування з представниками компанії вдалося сформулювати комплексне бачення того, яким має бути вебзастосунок як у візуальному, так і в технічному сенсі.

Замовник особливо наголосив на важливості створення візуального образу, що відповідав би архітектурній філософії студії – чіткому, мінімалістичному, з ідеальною гармонією кольору, простору та композиції. Такий підхід передбачає стриману, проте вишукану стилістику, де кожна деталь відіграє роль не лише естетичну, а й функціональну. Сайт мав стати цифровим відображенням підходів студії до проєктування: зосередження на якості, увазі до деталей, прагненні до ідеального балансу між формою та змістом.

Окрім зовнішнього вигляду, значну увагу було приділено інтуїтивності та зручності користувацького інтерфейсу. “Lukashev Architects” прагне створити для відвідувачів сайту комфортний, логічно організований простір, у якому будь-хто – незалежно від досвіду використання вебресурсів – зможе легко знайти потрібну інформацію. Вебзастосунок мав бути доступним з будь-яких пристроїв, незалежно від розміру екрана, з плавними анімаціями, зрозумілою навігацією та швидким відгуком на дії користувача. Замовник наголосив на важливості не лише зручної структури, а й приємного

емоційного досвіду – сайт має викликати довіру, зацікавлення, повагу до бренду.

Окрему увагу було приділено ключовим інформаційним розділам: головна сторінка повинна коротко презентувати основні цінності та філософію студії, розкривати перше враження про її підходи до архітектури та направляти користувача до важливих розділів сайту. Значну роль відіграє портфоліо, адже саме демонстрація вже реалізованих проєктів дозволяє потенційному замовнику оцінити рівень професіоналізму, стильову різноманітність і глибину архітектурного мислення команди. Було запропоновано реалізувати функцію повноекранного перегляду зображень, фільтрацію проєктів за типами або категоріями, щоб спростити пошук релевантного контенту. У розділі «Про студію» передбачено можливість розповісти більше про команду, принципи співпраці, досвід і цінності, які формують основу кожного проєкту. У свою чергу, форма зворотного зв’язку має забезпечити простий канал комунікації між студією та потенційним клієнтом, а також фіксувати отримані повідомлення у базі даних для подальшої обробки.

Замовник також висловив бажання мати на сайті розділ новин або блогу, де команда могла б публікувати статті, аналітику, професійні роздуми або розповідати про актуальні події з життя студії. Це не лише підтримує інформаційну активність бренду, а й дозволяє зміцнити експертну позицію компанії у сфері архітектури. У перспективі передбачено можливість розширення функціоналу вебзастосунку, зокрема додавання адміністративного інтерфейсу, що дозволить команді самостійно оновлювати контент без залучення розробників.

Технічна реалізація ресурсу спирається на використання сучасного, гнучкого та масштабованого стеку технологій. Інтерфейс будується з використанням React, що дозволяє реалізувати динамічні компоненти та забезпечити високу швидкість роботи. Серверна логіка реалізується за допомогою Node.js та Express, що надає змогу обробляти запити в реальному часі. У якості системи керування базами даних обрано PostgreSQL, що

дозволяє безпечно зберігати повідомлення користувачів, контент сторінок, технічні записи та іншу структуровану інформацію. Такий вибір технологій гарантує не лише стабільність і продуктивність, а й можливість масштабування проєкту в майбутньому.

Отже, аналіз вимог замовника засвідчив прагнення створити не просто сайт-візитівку, а комплексну цифрову платформу, яка б ефективно презентувала діяльність студії, забезпечувала зручну взаємодію з аудиторією, підтримувала імідж та допомагала будувати довготривалі комунікації з потенційними клієнтами. Ресурс має поєднувати у собі технічну досконаліть, витончену естетику та глибоку функціональність, ставши важливим інструментом у щоденній діяльності “Lukashev Architects”.

1.2 Формулювання вимог до системи

Під час розробки будь-якого програмного забезпечення надзвичайно важливо не лише окреслити загальні цілі, але й детально зафіксувати вимоги, які система повинна задовольняти. Це дозволяє структурувати процес створення застосунку, уникнути неузгодженостей та забезпечити результат, що відповідатиме очікуванням замовника. Вимоги, що ставляться до вебзастосунку для архітектурної студії, умовно поділяються на дві великі групи: функціональні та нефункціональні. Такий поділ дозволяє всебічно охопити як сутнісну частину роботи системи, так і її технічні й якісні характеристики.

Функціональні вимоги є першорядними з точки зору логіки взаємодії користувача з вебресурсом. Вони описують, які саме дії має дозволяти система, які дані відображати та які процеси реалізовувати. У випадку архітектурної студії, ключовим завданням є побудова такого інтерфейсу, що з одного боку презентує філософію, стиль і професійний рівень компанії, а з іншого – дозволяє потенційному клієнту швидко знайти необхідну

інформацію, ознайомитися з портфоліо, дізнатися більше про етапи співпраці та за потреби – зв’язатися з представниками бюро.

Структура вебсайту повинна забезпечити логічну й зрозумілу навігацію між основними розділами. Головна сторінка виконує роль першого контакту користувача з брендом – тут важливо коротко й виразно подати цінності компанії, продемонструвати стиль її робіт через візуальні матеріали та передбачити зручні переходи до детальніших підрозділів. Розділ «Про нас» має на меті не просто інформувати, а й викликати довіру, підкреслюючи людський вимір архітектури – з розповіддю про команду, історію майстерні, її бачення та методи роботи. Портфоліо, у свою чергу, є не лише вітриною проєктів, але й засобом переконання у професійності студії: проєкти повинні супроводжуватись описами, мати якісні зображення, можливість збільшення, а також фільтрацію за категоріями, напрямами чи масштабами. Додатковий розділ з інформаційними статтями дозволяє демонструвати експертність студії, а також забезпечує постійне оновлення контенту, що позитивно впливає на SEO-індексацію. І, нарешті, контактна форма має бути простою, зручною та ефективною – користувач повинен мати змогу залишити запит без складнощів, а студія – отримати необхідну інформацію для подальшого зв’язку.

Однак лише реалізація функцій – це лише частина завдання. Не менш важливими є нефункціональні вимоги, які стосуються якості реалізації всіх вищезазначених можливостей. Вони визначають, наскільки стабільно, швидко, безпечно й адаптивно працює вебзастосунок. В умовах сучасного цифрового середовища користувачі очікують, що сайт завантажиться практично миттєво, працюватиме плавно та коректно на будь-якому пристрої – від смартфона до великого монітора. Саме тому одним із ключових нефункціональних параметрів є швидкодія – кожна сторінка повинна відкриватися в межах 2-3 секунд, навіть за умов високого навантаження.

Окрема увага приділяється адаптивності інтерфейсу. Застосунок має бути оптимізований для перегляду з мобільних пристроїв, планшетів та

десктопів – з урахуванням різної роздільної здатності екранів, орієнтації пристрою та типу введення. Така гнучкість підвищує доступність ресурсу для широкої аудиторії та відповідає актуальним тенденціям UX-дизайну. Крім того, система повинна мати потенціал для подальшого масштабування – тобто її архітектура має бути побудована з урахуванням майбутніх оновлень, розширення функціоналу чи інтеграції нових технологій без потреби кардинального переписування коду.

Не менш важливим є забезпечення стабільності та безперервності роботи сайту. Навіть у разі відмови одного з компонентів система повинна залишатися доступною або мати механізми автоматичного відновлення. Це потребує впровадження резервних стратегій, надійного хостингу, а також регулярного моніторингу продуктивності. Крім того, у розробці слід врахувати питання безпеки – захист персональних даних користувачів, запобігання спробам несанкціонованого доступу, а також відповідність загальним вимогам GDPR або їхніх національних аналогів.

Підсумовуючи, варто зазначити, що формулювання вимог до вебзастосунку – це не просто технічне завдання, а комплексний процес, що поєднує у собі глибоке розуміння цілей замовника, очікувань користувачів і можливостей сучасних технологій. Лише поєднання чітко окресленої функціональності з високими нефункціональними стандартами забезпечує створення ефективного, зручного та надійного цифрового продукту, який успішно виконує як комунікативну, так і репутаційну функцію для архітектурної студії.

1.3 Вибір клієнтських технологій та побудова інтерфейсу

Під час створення сайту-візитки архітектурної майстерні було використано сучасний стек вебтехнологій, що охоплює HTML, CSS [17], JavaScript [18], а також бібліотеки й інструменти на базі React [16]. Такий вибір

дозволив забезпечити не лише естетично привабливий зовнішній вигляд, але й високу інтерактивність, швидкодію та зручність користування вебресурсом. Кожен компонент цієї системи відіграє важливу роль у формуванні швидкого, гнучкого та візуально привабливого вебінтерфейсу, адаптованого до потреб сучасного користувача.

HTML забезпечує базову структуру сторінки – формує розмітку всіх візуальних елементів: текстових блоків, зображень, заголовків, кнопок, форм введення [5]. Це своєрідний «скелет» будь-якого вебінтерфейсу, на основі якого будуються всі подальші елементи дизайну та логіки. У межах проєкту структура була логічно розділена на секції: головна сторінка, галерея реалізованих проєктів, інформація про архітектурну майстерню та контактна форма.

CSS відповідає за стилізацію цієї структури – задає кольори, шрифти, розміри, відступи, взаємне розташування елементів, а також адаптивну поведінку на різних розмірах екранів [5]. У цьому проєкті замість класичного CSS використовувався препроцесор SCSS, який розширює можливості стандартного CSS, дозволяючи використовувати змінні, вкладеність селекторів, міксіни, цикли та інші інструменти, що значно спрощують підтримку й масштабування стилів [14]. Наприклад, кольорова схема, що використовувалась на всіх сторінках, була винесена у змінні SCSS, що дозволяє швидко змінити загальний стиль сайту без переписування окремих стилів.

JavaScript додає інтерактивності до сторінки – дозволяє реагувати на дії користувача (кліки, наведення, заповнення форм), динамічно змінювати вміст без перезавантаження сторінки, реалізовувати анімації, підвантаження зображень або даних [5]. Під час розробки активно використовувалась DOM-модель (Document Object Model), що дозволяє отримувати доступ до HTML-елементів і змінювати їх властивості через JavaScript. Проте пряме маніпулювання DOM не завжди є ефективним рішенням, особливо коли сторінка постійно змінюється в реальному часі.

Саме тому для побудови інтерфейсу було обрано React – популярну бібліотеку JavaScript для створення динамічних вебзастосунків, що базується на компонентному підході [9]. Кожна частина інтерфейсу – шапка сайту, галерея, форма зв'язку – реалізована у вигляді окремого React-компонента, що полегшує повторне використання, тестування, зміну й масштабування елементів інтерфейсу.

Однією з ключових особливостей React є використання віртуального DOM (Virtual DOM). Замість того щоб одразу змінювати реальний DOM, React створює його копію в пам'яті, порівнює з попереднім станом і оновлює лише ті елементи, які дійсно змінилися. Це дозволяє значно підвищити продуктивність та зменшити навантаження на браузер. Такий підхід особливо ефективний при розробці інтерфейсів, що динамічно змінюються або оновлюються на основі дій користувача чи отриманих даних (див. рис. 1.1).

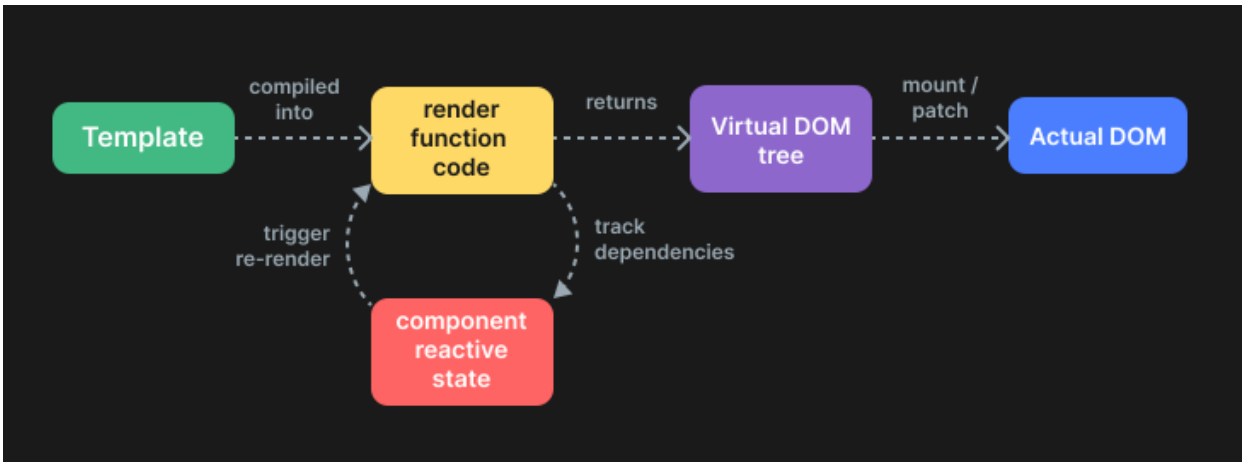


Рисунок 1.1 – Схематичне представлення роботи віртуального DOM

Для організації переходів між сторінками сайту використовувалась бібліотека React Router, яка дозволяє реалізувати маршрутизацію без перезавантаження сторінки [13]. Завдяки цьому користувач отримує досвід роботи з SPA (Single Page Application) – усі сторінки завантажуються миттєво, що створює відчуття безперервної навігації. Кожна сторінка – це окремий компонент, що підвантажується динамічно при переході, зберігаючи швидкість і ефективність.

Особлива увага приділялась реалізації контактної форми, яка є важливим елементом для комунікації з потенційними клієнтами. Для її створення були використані бібліотеки Formik і Yup. Formik забезпечує просте створення форм у React, автоматизує керування станом полів, обробку введення та надсилання [3]. У свою чергу, Yup дозволяє створити декларативну валідацію введених даних: перевірку email, мінімальної кількості символів у полі, обов’язковість заповнення тощо [15]. Такий підхід підвищує якість користувацького досвіду та дозволяє уникнути некоректних запитів.

Загалом використання вищезазначених інструментів дозволило створити гнучкий, адаптивний та зручний у використанні вебсайт з модульною архітектурою, яку легко підтримувати та розширювати. Такий підхід повністю відповідає сучасним вимогам до веброзробки – від чистоти коду та повторного використання компонентів до швидкості завантаження і високої взаємодії з користувачем. У майбутньому цю структуру легко масштабувати – наприклад, інтегрувавши бекенд API або додавши особистий кабінет адміністратора.

1.4 Серверне середовище Node.js та каркас Express.js у веброзробці

Після створення візуальної частини сайту архітектурної майстерні з використанням HTML, CSS і JavaScript, а також інтерактивних компонентів на базі React, наступним кроком у розробці сучасного вебзастосунку є впровадження серверної логіки. Хоча сайт виконує переважно презентаційну функцію – демонструє проєкти, описує діяльність майстерні та надає контактну інформацію – наявність серверної частини значно розширює можливості подальшого розвитку. Це включає підтримку обробки запитів із форми зворотного зв’язку, збереження повідомлень у базі даних, надсилання листів або навіть створення адміністративної панелі з керуванням контентом.

Для реалізації серверної логіки було обрано технологічний стек на основі Node.js [19] та Express.js, що ідеально підходить для невеликих, але масштабованих застосунків. Node.js – це середовище виконання JavaScript, яке працює поза межами браузера [7]. Основна його перевага – обробка великої кількості одночасних запитів без потреби створення нових потоків для кожного з них. Замість цього Node.js використовує модель з неблокуючим введенням-виведенням (non-blocking I/O) та асинхронну обробку подій через цикл подій (event loop), як показано на рисунку 1.2.

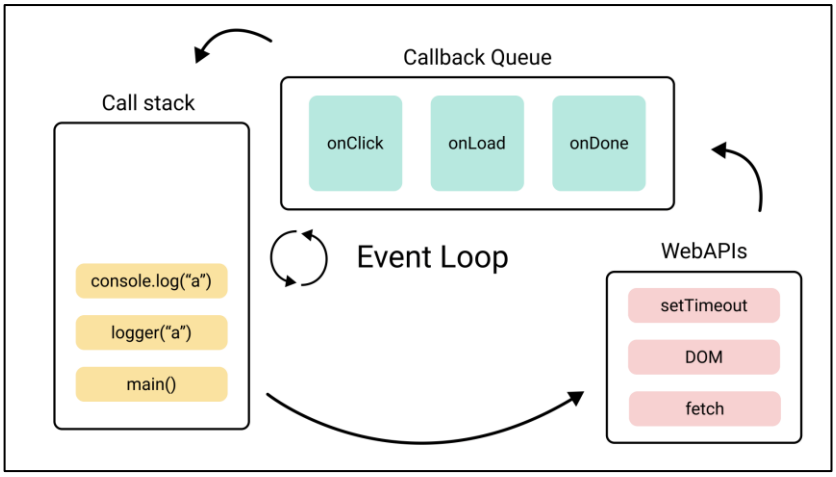


Рисунок 1.2 – Принцип роботи event loop

У поєднанні з Node.js застосовується Express.js – мінімалістичний вебфреймворк, який надає зручний API для організації маршрутизації, обробки запитів, формування відповідей та використання middleware [2]. Для сайту архітектурної студії Express.js може бути використаний для обробки POST-запитів із контактної форми, валідації введених даних, надсилання email-повідомлень, захисту від спаму (наприклад, із використанням Google ReCAPTCHA), а також для організації маршрутизації сторінок, таких як /about, /projects, /contact, тощо.

Додатковою перевагою цієї технологічної зв'язки є її масштабованість і розширюваність. Навіть якщо сайт на момент запуску не потребує складної бізнес-логіки, у майбутньому його легко адаптувати до нових потреб: додати базу даних (наприклад, MongoDB або PostgreSQL), реалізувати RESTful API

для інтеграції з мобільними застосунками, підключити систему авторизації та автентифікації користувачів, або створити панель керування для адміністратора.

На рисунку 1.3 наведено схему, що ілюструє взаємодію фронтенду, реалізованого на React, із серверною частиною на базі Node.js, Express.js та бази даних. Така структура забезпечує гнучкість і дозволяє реагувати на запити замовника в умовах змінного ринку.

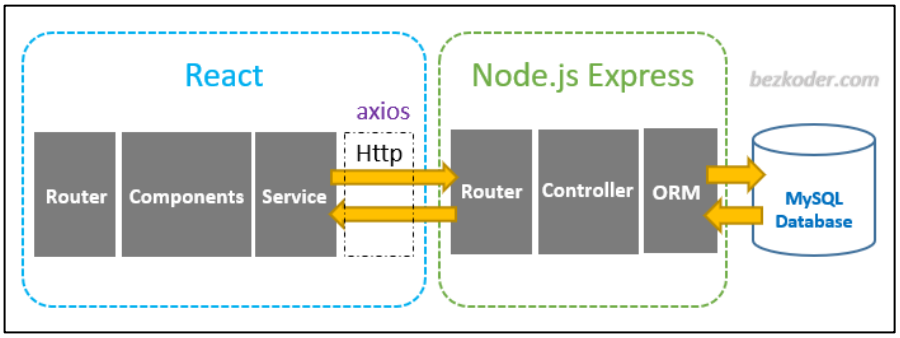


Рисунок 1.3 – Схема взаємодії фронтенду React із сервером на Node.js + Express.js + Database

Таким чином, реалізація серверної частини з використанням Node.js та Express.js є не лише доцільним, а й стратегічно виправданим рішенням. Воно дозволяє забезпечити надійну архітектуру застосунку, зберігаючи при цьому можливість масштабування та інтеграції нових функцій. Для архітектурної майстерні це означає, що сайт не залишиться лише статичним портфоліо, а може перетворитися на повноцінний цифровий інструмент комунікації, організації замовлень і взаємодії з клієнтами.

1.5 Реляційна база даних PostgreSQL та її переваги

Для реалізації надійної, гнучкої та масштабованої роботи з даними у вебзастосунках важливо обрати сучасну систему керування базами даних (СКБД), яка підтримує не лише базові операції зберігання й вибірки, а й

забезпечує розширені можливості обробки, захисту й оптимізації доступу до інформації. Однією з таких систем є PostgreSQL – потужна об’єктно-реляційна СКБД з відкритим кодом, яка поєднує у собі стандартизовану SQL-функціональність і гнучкість NoSQL-підходів [8].

PostgreSQL підтримує транзакції з ACID-гарантіями, що забезпечує цілісність і надійність даних навіть у випадку збоїв системи [20]. Серед додаткових можливостей – індекси, які значно прискорюють запити до великих обсягів даних, представлення (views) для створення віртуальних таблиць, збережені процедури та тригери для автоматизації бізнес-логіки на рівні бази даних. Розробник також може створювати власні типи даних, функції та оператори, що дозволяє гнучко адаптувати PostgreSQL під специфіку конкретного проєкту.

Особливістю PostgreSQL є її сувора відповідність стандарту SQL:2011, а також можливість виконання складних багаторівневих запитів, включаючи вкладені SELECT, агрегації, JOIN-операції, фільтрацію, угрупування та сортування. Вона має потужну підтримку транзакційної ізоляції, а завдяки механізму MVCC (мультиверсійного контролю конкурентного доступу) база дозволяє одночасно виконувати операції читання й запису без блокування таблиць, що особливо актуально для багатокористувацьких систем з високим навантаженням.

Завдяки підтримці JSON і JSONB, PostgreSQL дає змогу зберігати й обробляти напівструктуровані дані, подібно до NoSQL-баз, що ідеально підходить для зберігання налаштувань, метаданих або динамічних форм, отриманих з клієнтської частини. Така універсальність дозволяє PostgreSQL бути основою не тільки для класичних реляційних моделей, а й для гібридних сценаріїв роботи з даними.

У середовищі Node.js PostgreSQL активно використовується у зв’язці з ORM-бібліотеками, такими як Sequelize, TypeORM або Prisma, що забезпечує зручний інтерфейс для взаємодії з базою на рівні об’єктів, спрощує процес міграцій, управління зв’язками між таблицями, а також дозволяє ефективно

обробляти дані, що надходять із клієнта. Для більш контрольованої роботи часто застосовується також нативний клієнт pg, який надає повний доступ до SQL-функцій і дозволяє будувати запити з максимальною гнучкістю.

З точки зору безпеки, PostgreSQL підтримує автентифікацію за допомогою SSL, систему ролей і дозволів, а також надає можливість використання розширень для шифрування, аудиту дій користувачів, журналювання запитів. Для збереження даних і захисту від втрати підтримуються механізми реплікації (синхронної та асинхронної), а також інструменти резервного копіювання, як повного, так і інкрементального.

Активна міжнародна спільнота, регулярні оновлення, розширення можливостей через модулі (наприклад, PostGIS для геоданих або pgAudit для контролю доступу) роблять PostgreSQL одним із найкращих рішень для серверної частини будь-якого сучасного вебзастосунку. Для сайту архітектурної студії, який може з часом розвиватися від простої візитівки до повноцінного багатофункціонального порталу, PostgreSQL є логічним і надійним вибором.

1.6 Порівняльна характеристика альтернативних технологій

У сфері клієнтської розробки особливої уваги заслуговують такі інструменти, як React, Vue та Angular, кожен із яких має свої сильні сторони, філософію і сценарії використання. Безперечним лідером за популярністю на ринку є React – JavaScript-бібліотека з відкритим кодом, розроблена й підтримувана компанією Meta. React використовує компонентний підхід, що дозволяє легко масштабувати інтерфейс, повторно використовувати код, а також створювати динамічні, інтерактивні й продуктивні інтерфейси користувача. Особливістю React є використання віртуального DOM, що мінімізує витрати на оновлення сторінки й підвищує швидкість відгуку. Велика екосистема, активна спільнота, підтримка сучасних практик, таких як

хуки (hooks), контекст (context API), серверний рендеринг через Next.js – усе це робить React не просто бібліотекою, а цілою платформою для створення повноцінних фронтенд-рішень. Завдяки гнучкості й масштабованості React часто використовують не лише для вебзастосунків, а й для мобільної розробки через React Native.

Проте, попри домінування React, існує низка альтернатив, які можуть бути більш доцільними залежно від контексту проєкту. Однією з таких є Vue.js – прогресивний JavaScript-фреймворк, який вирізняється надзвичайно низьким порогом входу, зручною документацією та компактним, інтуїтивно зрозумілим ядром. Vue ідеально підходить для малих і середніх проєктів, MVP (мінімально життєздатних продуктів) або для команд із обмеженим досвідом у розробці фронтенду. Його синтаксис поєднує в собі найкращі практики з React та Angular, але при цьому Vue зберігає простоту та читабельність, що дозволяє швидко стартувати проєкт без потреби у складній конфігурації. Система реактивності у Vue базується на спостереженні за змінами стану, що забезпечує високу продуктивність без потреби в додатковому оптимізуванні. Крім того, Vue підтримує поступове впровадження у вже існуючі застосунки, що робить його хорошим вибором для модернізації застарілих проєктів або часткової реорганізації інтерфейсу.

Angular, у свою чергу, позиціонується як потужний повноцінний фреймворк із підходом «все-в-одному» (batteries included), який найкраще проявляє себе у великих і довготривалих проєктах, що потребують чіткої архітектури, суворої типізації та високої масштабованості. Angular базується на TypeScript, що вже на етапі розробки дозволяє виявляти помилки й забезпечує строгий контроль за типами даних, що важливо для безпеки та стабільності застосунку. У його складі вже є вбудовані засоби для форм, маршрутизації, роботи з HTTP-запитами, ін’єкції залежностей, юніт-тестування тощо – що зменшує залежність від сторонніх бібліотек і забезпечує єдність підходу в межах однієї команди. Це особливо важливо при роботі над масштабними корпоративними застосунками або в середовищах із жорсткими

вимогами до якості, документації та тестування. Однак крута крива навчання, суворі правила структурування проєкту й складна конфігурація середовища можуть стати бар'єром для новачків або команд, які потребують максимальної гнучкості.

Серверна частина сучасного вебзастосунку часто базується на Node.js – перевіреному часом середовищі, яке дозволяє використовувати одну мову (JavaScript/TypeScript) по всьому стеку. Завдяки подієво-орієнтованій моделі та неблокувальному вводу/виводу, Node добре підходить для застосунків із великою кількістю одночасних з'єднань. Новішою альтернативою виступає Deno – середовище від творця Node.js, що пропонує вбудовану підтримку TypeScript, відмову від node_modules, покращену безпеку через систему дозволів. Хоча екосистема Deno ще молода, він стрімко розвивається і стає перспективним варіантом для сучасних проєктів.

Серед серверних фреймворків Express.js зберігає популярність через простоту та гнучкість, однак у великих системах він іноді ускладнює масштабування. NestJS, натомість, базується на архітектурних принципах Angular і пропонує модульну структуру з підтримкою DI, декораторів та типізації – ідеально для складних корпоративних застосунків. Koa.js – ще один мінімалістичний варіант, розроблений командою Express, дозволяє повністю контролювати структуру проєкту, але вимагає більше досвіду.

З поширенням SSR та SSG значно зросла популярність повностекових фреймворків. Next.js (для React) – лідер у створенні SEO-оптимізованих застосунків, із потужною підтримкою гібридного рендерингу, API-роутів та оптимізації продуктивності. Nuxt.js (для Vue) забезпечує схожі можливості в середовищі Vue з акцентом на зручність та автоматизацію. Remix – сучасний фреймворк із фокусом на продуктивність і вебстандарти, а Astro дозволяє створювати надлегкі сайти, поєднуючи компоненти з різних фреймворків без зайвого JavaScript.

Для великих проєктів все більш актуальною стає архітектура мікрофронтендів, яка дозволяє командам незалежно розробляти окремі

частини інтерфейсу, навіть на різних технологіях. Завдяки Webpack Module Federation можлива динамічна інтеграція таких модулів без перевипуску основного застосунку, що прискорює оновлення та спрощує масштабування.

Узагальнюючи, можна сказати, що грамотний вибір і поєднання технологій відповідно до потреб проєкту, рівня підготовки команди та поставлених цілей дає змогу досягти оптимального результату – створити функціональний, масштабований і зручний у використанні вебзастосунок.

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ

2.1 Проєктування архітектури вебзастосунку

Проєктування архітектури вебзастосунку є визначальним етапом у створенні будь-якої сучасної інформаційної системи, оскільки саме на цьому етапі закладається логіка взаємодії між усіма складовими застосунку, визначається технологічний стек, принципи масштабування, безпеки та обслуговування. У межах дипломного проєкту, присвяченого розробці вебзастосунку-візитки для архітектурної майстерні, обрано трирівневу архітектуру, яка включає клієнтську частину, серверну частину та базу даних. Такий підхід забезпечує чітке розділення відповідальностей, спрощує підтримку коду та дозволяє масштабувати застосунок у разі потреби.

Клієнтська частина реалізована з використанням бібліотеки React, що дозволяє створювати гнучкий, адаптивний та інтуїтивно зрозумілий інтерфейс. Компонентний підхід, на якому базується React, забезпечує модульність, повторне використання коду та легке оновлення окремих елементів без потреби переписувати інші частини інтерфейсу. Важливу роль відіграє форма зворотного зв’язку, яка створена за допомогою бібліотек Formik та Yup для зручної роботи з формами та їх валідацією. Для забезпечення захисту від спаму застосовується Google ReCAPTCHA v3, інтегрована безпосередньо в процес надсилання повідомлення, що не створює зайвих перешкод для користувача [4]. Вся логіка взаємодії з сервером реалізована за допомогою fetch-запитів, які дозволяють надсилати дані форми у форматі JSON до серверної частини.

Серверна частина побудована на основі Node.js у поєднанні з фреймворком Express. Така комбінація забезпечує ефективну обробку HTTP-запитів, реалізацію бізнес-логіки, інтеграцію з базою даних та зовнішніми сервісами, зокрема Google ReCAPTCHA. У відповідь на запити з клієнта сервер проводить перевірку на коректність даних, виконує валідацію

ReCAPTCHA-токена шляхом надсилання запиту до відповідного API Google, а після успішної перевірки – зберігає отриману інформацію в базі даних. Обробка запитів відбувається із врахуванням базових принципів безпеки, включаючи обробку помилок і запобігання SQL-ін’єкціям. Уся логіка доступу до даних реалізована вручну з використанням бібліотеки pg, що дозволяє повністю контролювати запити до бази та уникати небажаних автоматичних обгорток, характерних для ORM-рішень.

Для збереження даних у проєкті використовується PostgreSQL – сучасна реляційна система управління базами даних, яка відзначається надійністю, масштабованістю та підтримкою складних запитів. У рамках вебзастосунку база даних виконує функцію централізованого сховища інформації, зокрема повідомлень, що надходять із форми зворотного зв’язку. Збереження таких даних, як ім’я відправника, його електронна адреса, текст повідомлення та дата надсилання, дозволяє надалі аналізувати комунікацію з потенційними клієнтами, а також слугує джерелом для побудови розширеної статистики або додаткових адміністративних функцій у майбутньому. Структура таблиць у базі даних спроєктована з урахуванням можливості подальшого розширення функціональності застосунку, наприклад, додавання окремих блоків із новинами, прикладами проєктів або галереями.

Загальна архітектура вебзастосунку забезпечує послідовну та безпечну взаємодію між усіма рівнями. Користувач взаємодіє лише з інтерфейсом, не маючи прямого доступу до логіки обробки чи до бази даних, що відповідає принципам захисту інформації. Серверна частина виконує роль шлюзу, перевіряє та обробляє запити, гарантуючи, що до бази надходять лише валідні дані. Завдяки чіткому розмежуванню обов’язків між клієнтом, сервером та сховищем даних досягається висока гнучкість системи та можливість її подальшої адаптації до нових бізнес-завдань.

На рисунку 2.1 представлено загальну архітектурну схему вебзастосунку, де відображено логіку взаємодії між клієнтським інтерфейсом, сервером і базою даних.

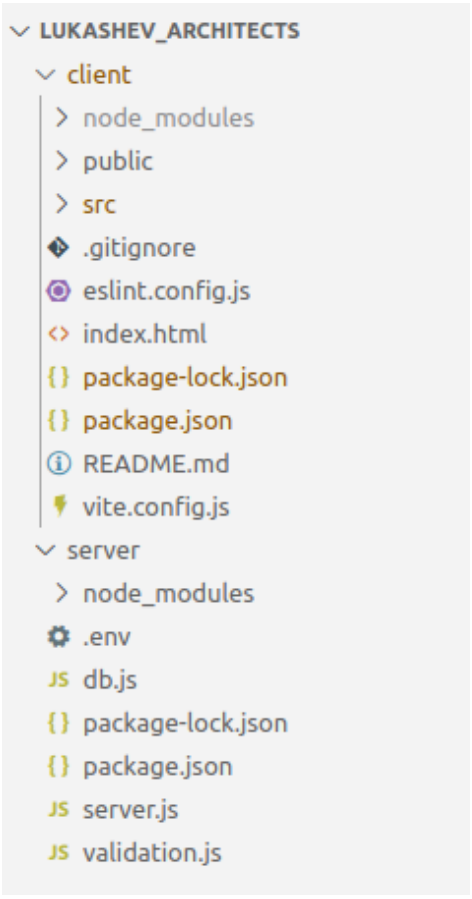


Рисунок 2.1 – Архітектура вебзастосунку з React, Node.js та PostgreSQL

Така візуалізація дозволяє краще зрозуміти функціональне розмежування рівнів та шляхи передачі даних у системі.

2.2 Проєктування інтерфейсу користувача

Проєктування інтерфейсу користувача (UI) є однією з найважливіших складових при розробці вебзастосунку для архітектурної майстерні, оскільки саме він є основним каналом комунікації між користувачем і системою. Якість інтерфейсу, його зручність та привабливість безпосередньо впливають на враження користувачів від сайту, ефективність навігації, швидкість виконання цільових дій та загальний досвід використання ресурсу. У випадку з вебзастосунком архітектурної майстерні, важливим є не тільки забезпечення зручного функціоналу, але й передача через інтерфейс індивідуальності

бренду студії. Створення гармонійного, естетично привабливого інтерфейсу, що відображає філософію студії, є ключовим аспектом розробки.

Основна мета, яка ставилася перед командою розробників, полягала в тому, щоб створити чистий, мінімалістичний інтерфейс, який одночасно є інтуїтивно зрозумілим і зручним для користувачів, незалежно від пристрою. Враховуючи швидкий розвиток мобільних технологій, було вирішено зробити вебзастосунок адаптивним, щоб він коректно відображався як на стаціонарних комп’ютерах, так і на мобільних пристроях, зберігаючи функціональність та естетичний вигляд.

У процесі проєктування було використано компонентний підхід, що є типовим для розробки на React [10]. Це дозволяє створювати самодостатні компоненти, які можуть бути багаторазово використані на різних сторінках, забезпечуючи таким чином масштабованість і спрощуючи підтримку коду. Структура застосунку була розділена на сторінки (pages) та компоненти (components), що дає змогу легко оновлювати або змінювати окремі частини інтерфейсу без необхідності переписувати великий обсяг коду.

Розглянемо **основні сторінки** вебзастосунку.

Головна сторінка (HomePage) – це перше враження, яке користувач отримує про студію. Вона не лише інформує про основні аспекти діяльності студії, а й візуально презентує її стиль через демонстрацію проєктів у галереї. Окрім цього, на головній сторінці користувачі можуть знайти коротке ознайомлення з командою та цінностями студії.

Про нас (AboutPage) – ця сторінка дає більше деталей про історію створення студії, її команду, а також принципи роботи. Вона покликана створювати атмосферу довіри та познайомити відвідувачів з корпоративною культурою.

Архітектурні проєкти (ArchitecturePage) – сторінка, що служить портфоліо для демонстрації виконаних проєктів. Тут представлено зображення, описи та технічні характеристики кожного проєкту, що дозволяє потенційним замовникам оцінити рівень майстерності студії.

Контакти (ContactPage) – сторінка зі зручною формою зворотного зв’язку та інтегрованою картою. Це дозволяє користувачам не лише надіслати повідомлення, але й швидко знайти адресу студії, номер телефону чи інші способи зв’язку.

Інформаційні сторінки (InfoPage, WorkProcessPage) – ці сторінки детально описують процес співпраці з майстернею, етапи розробки проєктів, відповідальність студії та особливості підходу до клієнтів.

На рисунку 2.2 представлена схема структури клієнтської частини вебзастосунку, що показує взаємозв’язки між сторінками та компонентами.

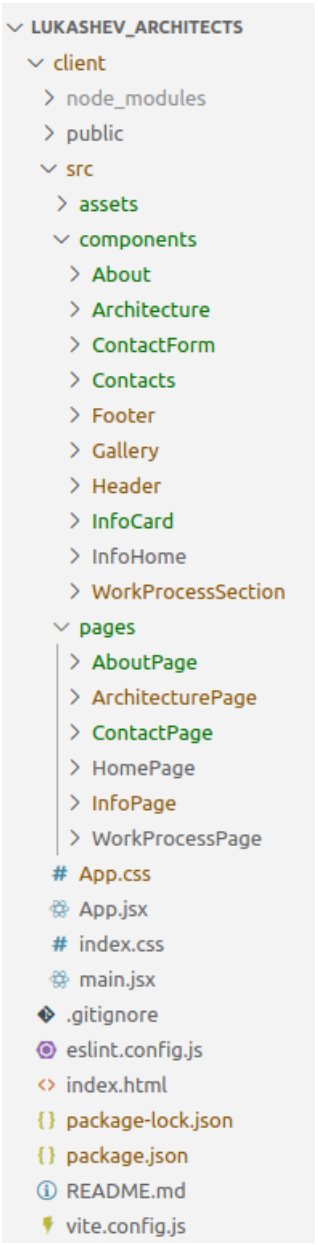


Рисунок 2.2 – Структура клієнтської частини вебзастосунку

Всі сторінки побудовані з використанням **повторно застосовуваних компонентів**, що дозволяє створювати більш гнучку архітектуру та економити час на розробку:

- **header** – компонент, що містить навігаційне меню, яке адаптується до мобільних пристроїв через бургер-меню (цей компонент відповідає за організацію навігації між різними частинами сайту);
- **footer** – містить важливу інформацію, таку як контактні дані, авторські права, посилання на соціальні мережі;
- **gallery** – використовуються для виведення портфоліо та проєктних робіт;
- **contactForm** – компонент для реалізації форми зворотного зв’язку, який включає валідацію введених даних та інтеграцію з Google reCAPTCHA для захисту від спаму.

Ключові принципи розробки інтерфейсу:

- **адаптивність** – використання SCSS для створення медіа-запитів забезпечує коректне відображення вебзастосунку на різних типах екранів, що дозволяє користувачам мати однаково зручний досвід незалежно від пристрою;
- **кросбраузерність** – тестування інтерфейсу на різних браузерах гарантує стабільну роботу сайту у всіх популярних браузерах, що важливо для досягнення високої доступності;
- **єдине стильове оформлення** – стилі застосунку винесені в окремі файли, що дозволяє централізовано керувати усіма візуальними елементами, такими як кольорова схема, шрифти та розміри елементів;
- **висока продуктивність** – для забезпечення швидкості роботи сайту використовувався інструмент Vite, що дозволяє оптимізувати процес збірки та прискорює завантаження сторінок, що важливо для поліпшення досвіду користувачів.

Навігація також була ретельно продумана. Використання **React Router** дозволило організувати безшовний перехід між сторінками без

перезавантаження, що створює плавний та інтуїтивно зрозумілий досвід для користувача. Це також позитивно впливає на продуктивність сайту та знижує навантаження на сервер.

Проєктування інтерфейсу базувалося на принципах сучасного вебдизайну, орієнтуючись на потреби цільової аудиторії – потенційних замовників архітектурних послуг. Структура сайту спрощена, а всі основні функції доступні в кілька кліків, що забезпечує зручність і логічність навігації.

2.3 Налаштування серверної частини вебзастосунку

Серверна частина вебзастосунку реалізована з використанням середовища виконання Node.js у поєднанні з фреймворком Express.js, що надає потужні інструменти для обробки HTTP-запитів, маршрутизації, підключення middleware та реалізації REST API. Такий вибір технологій зумовлений їхньою простотою у використанні, активною спільнотою, широким спектром доступних бібліотек та високою продуктивністю, що робить їх придатними для побудови сучасних масштабованих вебсервісів.

Основна функція серверної частини полягає у забезпеченні зв’язку між клієнтською частиною та базою даних, обробці вхідних запитів, виконанні валідації та поверненні відповідей відповідно до бізнес-логіки застосунку. На сервері реалізовано всі ключові аспекти взаємодії: отримання даних з форми зворотного зв’язку, перевірка коректності цих даних, запис у базу даних та обробка можливих помилок, що можуть виникнути під час виконання запиту.

Файлова структура серверної частини має логічно поділену архітектуру. У директорії server зосереджено основні модулі, що забезпечують незалежну роботу кожного елемента:

– server.js – головний файл, у якому налаштовується запуск сервера, підключення до бази даних, CORS, middleware для обробки JSON-запитів, а також реалізується маршрутизація;

– db.js – модуль, відповідальний за підключення до бази даних PostgreSQL з використанням бібліотеки pg і створенням пулу з'єднань для оптимізації виконання SQL-запитів;
– validation.js – файл, у якому описано правила перевірки вхідних даних: обов'язковість полів (name, email, phone), перевірка на правильність формату email тощо;
– .env – файл середовищних змінних, що містить чутливі дані, такі як адреса сервера БД, логін і пароль, і ніколи не включається до репозиторію завдяки .gitignore.

Зразок конфігурації файлу .env наведено на рисунку 2.3.



Рисунок 2.3 – Налаштування файлу ".env"

Окрему увагу приділено безпеці збереження конфіденційних даних. Файл .env дозволяє зберігати інформацію про підключення до PostgreSQL без розголошення її у відкритому коді. Значення з цього файлу використовуються через пакет dotenv, що автоматично підвантажує змінні середовища на момент запуску сервера. У db.js створюється пул підключень, що забезпечує ефективний розподіл запитів до таблиці contacts, де зберігаються надіслані користувачем дані.

У файлі server.js реалізовано запуск сервера на обраному порту, зазвичай 3001. Після запуску сервер починає слухати запити з клієнтської частини. Для забезпечення крос-доменної взаємодії використовується бібліотека cors, що дозволяє приймати запити з домену клієнтського інтерфейсу. Крім того, використовується express.json() як middleware для обробки JSON-форматів, що дозволяє без проблем отримувати дані з тіла POST-запитів.

Маршрут /contact призначений для прийому повідомлень користувача. На цьому маршруті реалізовано ланцюжок middleware, що спочатку проводить

валідацію даних, а потім, у разі їхньої коректності, записує їх до бази даних. У випадку помилок валідації користувач отримує відповідь з кодом 400, тоді як внутрішні помилки сервера повертаються з кодом 500. Для зручності налагодження передбачена також система логування повідомлень про помилки.

Структуру проєкту з файлами та прикладом коду реалізації подано на рисунку 2.4.

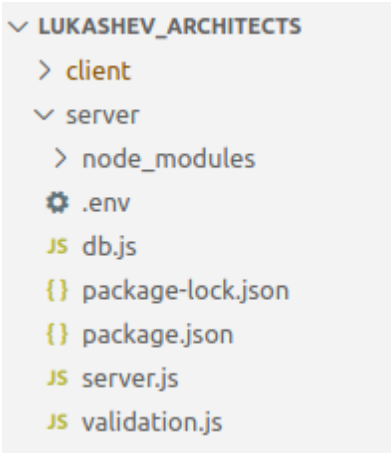


Рисунок 2.4 – Структура та код серверної частини проєкту (Node.js + Express + PostgreSQL)

Важливою складовою є захист від шкідливих або некоректних запитів. Перед збереженням даних до бази перевіряються всі обов’язкові поля, що унеможливлює ін’єкції або збереження порожніх значень. У перспективі можна реалізувати додаткові шари безпеки – зокрема, *rate-limiting* для захисту від DDoS, а також *CSRF*-захист для більш безпечної роботи з формами.

Таким чином, налаштування серверної частини не лише забезпечує стабільне функціонування вебзастосунку, а й створює гнучку та безпечну основу для масштабування. Враховуючи чітке розділення обов’язків між модулями, легко додавати нові маршрути, змінювати логіку обробки або розширювати функціональність, наприклад, через додавання нових типів запитів, розгортання користувацької автентифікації або інтеграції з зовнішніми сервісами.

2.4 Створення ER-діаграми бази даних

Для реалізації серверної логіки вебзастосунку, пов’язаної зі збором та зберіганням повідомлень користувачів, було створено базу даних з назвою lukashevarchitects. Основною її функцією є безпечне та надійне зберігання інформації, отриманої через контактну форму. З огляду на вузьку спеціалізацію вебзастосунку (збір звернень та контактних даних), структура бази даних є відносно простою, однак водночас відповідає усім вимогам щодо збереження персональних даних, цілісності інформації та подальшого масштабування.

У межах бази даних реалізовано одну основну сутність – таблицю contacts, яка зберігає повідомлення користувачів. Такий підхід дозволяє централізовано обробляти всі звернення, що надходять із клієнтської частини застосунку, включно з ім’ям, електронною поштою, текстом повідомлення, а також підтвердженням ознайомлення з політикою конфіденційності. Незважаючи на те, що на початковому етапі структура бази є односутнісною, вона передбачає можливість подальшого доповнення новими таблицями, наприклад для збереження інформації про обробку звернень, користувачів адміністративної панелі або логів системи.

Система керування базами даних (СКБД) обрана на користь **PostgreSQL**, оскільки вона забезпечує:

– високу продуктивність при роботі зі структурованими даними;
– підтримку транзакцій і надійне збереження даних;
– сумісність із Node.js через офіційний пакет pg;
– безпеку з’єднань і можливість гнучкого керування правами доступу;
– підтримку складних типів даних, індексації та тригерів.

Таблиця contacts розроблена з урахуванням типових вимог до контактних форм і включає такі поля, як ім’я, електронна адреса, текст повідомлення, номер телефону, дата відправлення та підтвердження ознайомлення з політикою конфіденційності. Детальна структура наведена в таблиці 2.1.

Таблиця 2.1 – Зберігання контактних даних і повідомлень користувачів

Назва поля	Тип даних	Призначення
id (PK)	SERIAL	Унікальний ідентифікатор запису. Автоматично генерується як первинний ключ.
name	VARCHAR(100) NOT NULL	Ім’я користувача. Обов’язкове для заповнення.
email	VARCHAR(150) NOT NULL	Електронна пошта користувача. Також є обов’язковим полем.
phone	VARCHAR(20)	Номер телефону. Не є обов’язковим, але може бути заповнений користувачем.
message	TEXT	Основне текстове повідомлення користувача.
policy	BOOLEAN NOT NULL	Логічне поле, що вказує на згоду з політикою конфіденційності.
recaptcha	TEXT	Токен перевірки Google reCAPTCHA для захисту від ботів.
created_at	TIMESTAMP DEFAULT CURRENT_TIMESTAMP	Дата та час створення запису. Встановлюється автоматично при надсиланні форми.

Таке проєктування забезпечує збереження всіх необхідних даних для ефективної роботи контактної форми, дотримуючись принципів нормалізації даних, мінімізації надлишковості та забезпечення цілісності.

Візуалізацію цієї структури відображає ER-діаграма на рисунку 2.5, яка дозволяє краще зрозуміти взаємозв’язки між атрибутами таблиці.

ER-діаграма (Entity-Relationship diagram) відображає структуру бази даних та зв’язки між сутностями. У поточному випадку вона включає лише одну таблицю – contacts, тож діаграма ілюструє поля цієї таблиці, типи даних та ключові атрибути. Поле id виступає первинним ключем, що забезпечує унікальність записів.

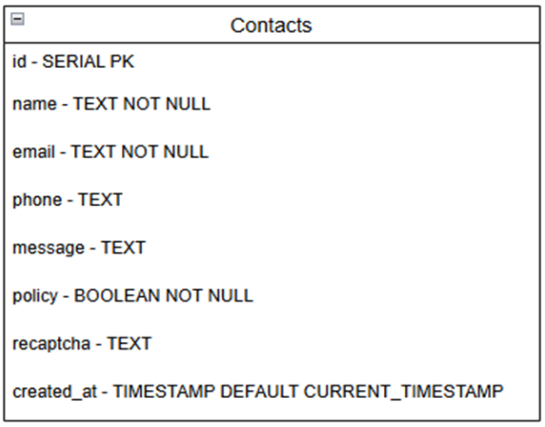


Рисунок 2.5 – ER-діаграма таблиці contacts

Попри мінімалістичну структуру, ця модель є повнофункціональною для потреб вебзастосунку. У майбутньому до ER-діаграми можуть бути додані додаткові таблиці:

– admin_users – для автентифікації адміністраторів;
– responses – для фіксації відповідей чи приміток;
– logs – для моніторингу дій користувачів та системних подій.

Особливу увагу приділено захисту даних. Використання Google reCAPTCHA допомагає запобігти спаму, а перевірка обов’язкових полів (name, email, policy) і валідація на обох сторонах знижують ризики некоректних або шкідливих запитів. Хоч таблиця не містить чутливих даних, для продакшн-середовища передбачено обмеження доступу до БД і шифрування з’єднань.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКА

3.1 Розробка фронтенду з використанням React.js

Фронтенд частина вебзастосунку була реалізована з використанням бібліотеки React.js, що дало змогу створити динамічний і адаптивний інтерфейс із чітким поділом на компоненти. Основний візуальний стиль базується на комбінації чорного та білого кольорів – чорний фон, контрастні білі тексти та лаконічні графічні елементи, що підкреслюють архітектурний характер сайту. Такий підхід забезпечує сучасний вигляд застосунку, високий рівень читабельності та акцент на візуальному контенті.

На початковому етапі були створені універсальні компоненти – Header (шапка сайту) і Footer (підвал), які використовуються на всіх сторінках. Їх впровадження дозволило підтримувати візуальну цілісність інтерфейсу, спростити структурування коду та повторне використання. Компонент Header містить логотип, навігаційне меню та адаптивний блок для мобільних пристроїв і завжди розташовується у верхній частині сторінки (див. рис. 3.1).



Рисунок 3.1 – Вигляд компонента Header у вебінтерфейсі

Компонент Footer, навпаки, розміщується внизу сторінки й містить контактну інформацію, копірайт і посилання на соціальні мережі. Його оформлення витримане в мінімалістичному стилі (див. рис. 3.2).



Рисунок 3.2 – Вигляд компонента Footer у вебінтерфейсі

Уся структура інтерфейсу побудована за принципом композиції – сторінки створюються шляхом об’єднання окремих компонентів. Наприклад, головна сторінка включає Header, Gallery, InfoHome та Footer (див. рис. 3.3).

```
import React from 'react'
import Header from '../../components/Header'
import Footer from '../../components/Footer'
import Gallery from '../../components/Gallery'
import InfoHome from '../../components/InfoHome'

const HomePage = () => {
  return (
    <div>
      <Header></Header>
      <Gallery></Gallery>
      <InfoHome></InfoHome>
      <Footer></Footer>
    </div>
  )
}

export default HomePage
```

Рисунок 3.3 – Структура головної сторінки з компонентів

Компоненти реалізовані з використанням функціонального підходу: хуки [12], props [11], SCSS-модулі для стилів, що забезпечує чистий та ефективний код (див. рис. 3.4, 3.5).

```
import { NavLink } from 'react-router-dom'
import React, { useState } from 'react'
import styles from './Header.module.scss'
import logo from '/src/assets/images/Logo/photo_2025-04-03_14-42-14.jpg'

const Header = () => {
  const [menuOpen, setMenuOpen] = useState(false)

  return (
    <header className={styles.header}>
      <div className={styles.container}>
        <NavLink to='/' className={styles.logoContainer}>
          <img src={logo} className={styles.logo} alt='Logo' />
        </NavLink>

        {/* Бургер-кнопка */}
        <div className={styles.burger} onClick={() => setMenuOpen(!menuOpen)}>
          <span className={menuOpen ? styles.open : ''}></span>
          <span className={menuOpen ? styles.open : ''}></span>
          <span className={menuOpen ? styles.open : ''}></span>
        </div>
```

Рисунок 3.4 – Код компонента Header у середовищі розробки

```
import React from 'react'
import { Link } from 'react-router-dom'
import {
  FaMapMarkerAlt,
  FaPhoneAlt,
  FaEnvelope,
  FaYoutube,
  FaInstagram
} from 'react-icons/fa'
import styles from './Footer.module.scss'
import WhiteLogo from '/src/assets/images/Logo/Logo_LA_PNG_white.png'

const Footer = () => {
  return (
    <section className={styles.footerContainer}>
      <ul className={styles.footerList}>
        <li className={styles.whiteLogoContainer}>
          <img src={WhiteLogo} alt='White Logo' className={styles.whiteLogo} />
        </li>
        <li className={styles.footerItem}>
          <h5 className={styles.footerTitle}>Portfolio</h5>
          <Link to='/architecture' className={styles.footerLink}>
            Architecture
          </Link>
          <Link to='/work_process' className={styles.footerLink}>
            Work Process
          </Link>
        </li>
```

Рисунок 3.5 – Код компонента Footer у середовищі розробки

Підключення компонентів до сторінок відбувається шляхом їх імпорту та використання JSX-розмітки. Наприклад, сторінка зворотного зв’язку відображає послідовно: Header, основний контент (форму), Footer (див. рис. 3.6).

```
import React from 'react'
import Header from '../../components/Header'
import Footer from '../../components/Footer'
import ContactForm from '../../components/ContactForm'
import Contacts from '../../components/Contacts'
import styles from './ContactPageStyles.module.scss'

const ContactPage = () => {
  return (
    <>
      <Header></Header>
      <div className={styles.ContactsContainer}>
        <Contacts></Contacts>
        <ContactForm></ContactForm>
      </div>
      <Footer></Footer>
    </>
  )
}

export default ContactPage
```

Рисунок 3.6 – Складання сторінки з компонентів у коді

Одним із прикладів реалізації є компонент ContactForm, що містить поля для імені, пошти, телефону, повідомлення, чекбокс згоди з політикою конфіденційності та Google reCAPTCHA. Управління формою виконується через Formik і Yup для валідації (див. рис. 3.7).

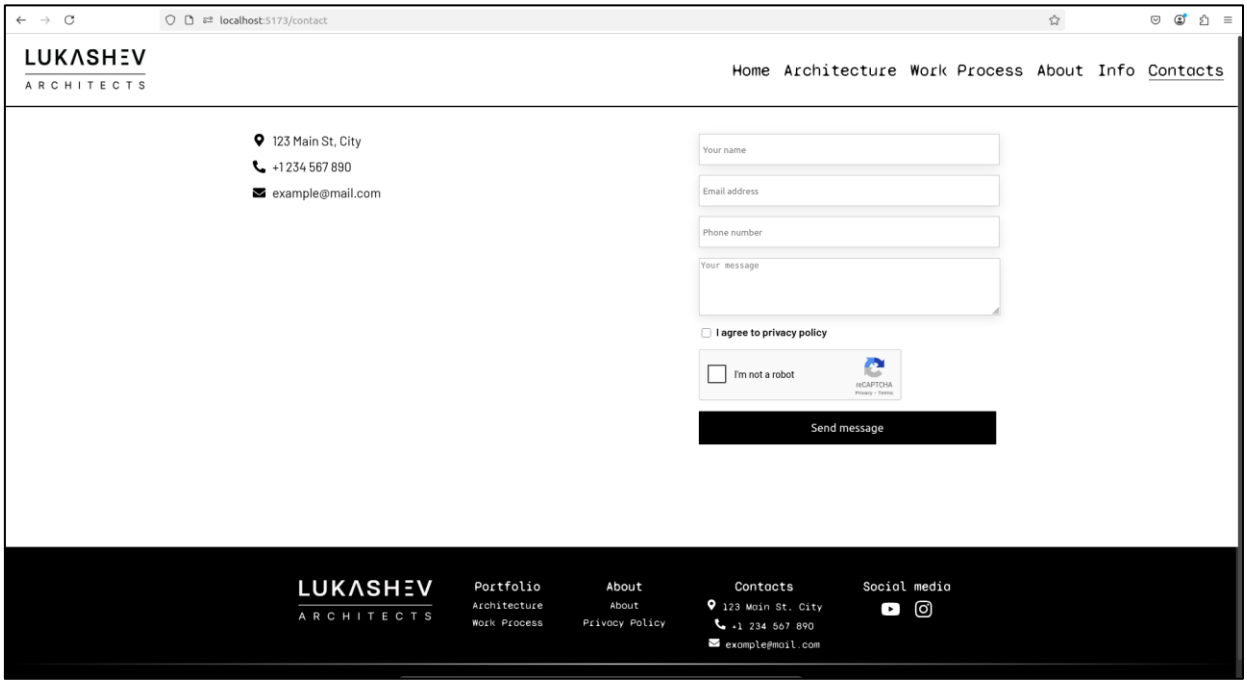


Рисунок 3.7 – Вигляд сторінки з формою зворотного зв’язку у вебінтерфейсі

Кожна сторінка створюється за єдиною логікою: спочатку Header, потім унікальний контент, далі Footer. Наприклад, сторінка «Про нас» (AboutPage) містить компоненти About, Gallery, і завершується Footer (див. рис. 3.8).

```
import React from 'react'
import Header from '../../components/Header'
import About from '../../components/About'
import Gallery from '../../components/Gallery'
import Footer from '../../components/Footer'
import styles from './AboutPageStyles.module.scss'

const AboutPage = () => {
  return (
    <div className={styles.aboutPageContainer}>
      <Header></Header>
      <About></About>
      <Gallery></Gallery>
      <Footer></Footer>
    </div>
  )
}

export default AboutPage
```

Рисунок 3.8 – Структура сторінки “Про нас” у коді

А сторінка «Архітектурні проєкти» (ArchitecturePage) виводить проєкти студії у вигляді галереї карток, використовуючи компонент Architecture, який приймає дані про об’єкти (див. рис. 3.9, 3.10).

```
import React from 'react'
import Header from '../../components/Header'
import Gallery from '../../components/Gallery'
import Footer from '../../components/Footer'
import Architecture from '../../components/Architecture'

const ArchitecturePage = () => {
  return (
    <div>
      <Header></Header>
      <Gallery></Gallery>
      <Architecture></Architecture>
      <Footer></Footer>
    </div>
  )
}

export default ArchitecturePage
```

Рисунок 3.9 – Код сторінки “Архітектурні проєкти” у середовищі розробки



Рисунок 3.10 – Вигляд портфоліо проєктів у вебінтерфейсі

Інформаційні сторінки, такі як InfoPage та WorkProcessPage, містять текстовий контент і структуровані секції. Наприклад, WorkProcessPage включає WorkProcessSection, який показує поетапний процес співпраці (див. рис. 3.11, 3.12).

```
import React from 'react'
import Header from '../../components/Header'
import Footer from '../../components/Footer'
import Gallery from '../../components/Gallery'
import WorkProccesSection from '../../components/WorkProcessSection'

const WorkProccesPage = () => {
  return (
    <>
      <Header></Header>
      <WorkProccesSection></WorkProccesSection>
      <Gallery></Gallery>
      <Footer></Footer>
    </>
  )
}

export default WorkProccesPage
```

Рисунок 3.11 – Структура інформаційної сторінки WorkProcessPage



Рисунок 3.12 – Вигляд секції процесу роботи в інтерфейсі сайту

Усі компоненти адаптивні: інтерфейс автоматично перебудовується залежно від пристрою. Наприклад, Header на мобільних пристроях перетворюється на бургер-меню, а галереї пристосовуються до вертикального скролінгу (див. рис. 3.13).

Навігація між сторінками реалізована через **React Router**, що дозволяє переходити без перезавантаження сторінки. Кожен маршрут відповідає окремому компоненту (див. рис. 3.14).

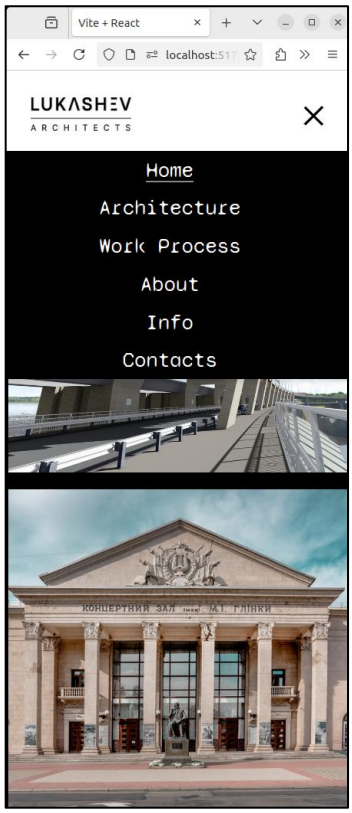


Рисунок 3.13 – Мобільна версія головної сторінки

```
import { BrowserRouter as Router, Route, Routes, Link } from 'react-router-dom'
import './App.css'
import HomePage from './pages/HomePage'
import ArchitecturePage from './pages/ArchitecturePage'
import WorkProcessPage from './pages/WorkProcessPage'
import AboutPage from './pages/AboutPage'
import InfoPage from './pages/InfoPage'
import ContactPage from './pages/ContactPage'

function App () {
  return (
    <>
      <Router>
        <Routes>
          <Route path='/' element={<HomePage />} />
          <Route path='/architecture' element={<ArchitecturePage />} />
          <Route path='/work_process' element={<WorkProcessPage />} />
          <Route path='/about' element={<AboutPage />} />
          <Route path='/info' element={<InfoPage />} />
          <Route path='/contact' element={<ContactPage />} />
        </Routes>
      </Router>
    </>
  )
}

export default App
```

Рисунок 3.14 – Приклад маршрутизації сторінок у React Router

Для організації стилів застосовано SCSS-модулі, які дозволяють створювати ізольовані стилі для кожного компонента [1]. Це мінімізує ймовірність конфлікту між стилями та забезпечує чисту структуру проєкту.

У підсумку, фронтенд вебзастосунку було реалізовано з дотриманням сучасних підходів до розробки інтерфейсів – з увагою до структури коду, повторного використання компонентів, адаптивності й зручності користування.

3.2 Розробка бекенду з використанням Node.js

Серверна частина вебзастосунку була розміщена в окремій директорії server, яка містить усі необхідні конфігураційні та програмні файли для обробки запитів, валідації даних, взаємодії з базою даних, а також забезпечення безпеки з'єднання. Структура цієї частини проєкту представлена у вигляді дерева файлів у середовищі розробки (див. рис. 3.15).

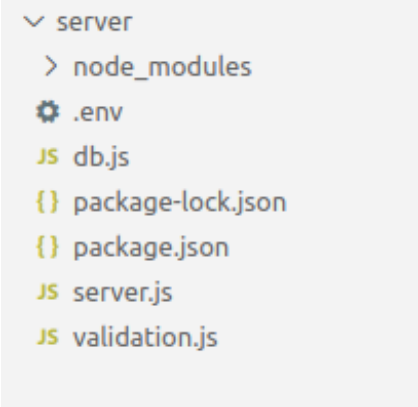


Рисунок 3.15 – Структура серверної частини проєкту у середовищі розробки

Для розгортання серверної частини було обрано JavaScript-платформу Node.js разом із фреймворком Express.js, який забезпечує простий та гнучкий інтерфейс для створення REST API [6]. Головний файл, з якого починається виконання застосунку, має назву server.js. Саме в ньому ініціалізується додаток, підключаються необхідні модулі (express, cors, dotenv, pg, а також схема валідації), а також налаштовується порт прослуховування запитів (див. рис. 3.16).

```
const express = require('express')
const cors = require('cors')
require('dotenv').config()

const pool = require('./db')
const contactSchema = require('./validation')

const app = express()
const PORT = process.env.PORT || 3001

app.use(cors({ origin: 'http://localhost:5173' }))
app.use(express.json())

app.post('/contact', async (req, res) => {
  try {
    const validated = await contactSchema.validate(req.body)

    const { name, email, phone, message, policy, recaptcha } = validated

    await pool.query(
      `INSERT INTO contacts (name, email, phone, message, policy, recaptcha)
        VALUES ($1, $2, $3, $4, $5, $6)`,
      [name, email, phone, message, policy, recaptcha]
    )

    res.status(201).json({ success: true, message: 'Сообщение сохранено' })
  } catch (error) {
    if (error.name === 'ValidationError') {
      return res.status(400).json({ error: error.message })
    }

    console.error('Ошибка:', error)
    res.status(500).json({ error: 'Ошибка сервера' })
  }
})

app.listen(PORT, () => {
  console.log(`Сервер работает на http://localhost:${PORT}`)
})
```

Рисунок 3.16 – Основний серверний файл (server.js) з реалізацією маршруту POST /contact

У процесі конфігурації сервер було налаштовано на прийом запитів лише з дозволеного джерела – домену клієнтської частини (http://localhost:5173), що гарантує дотримання політики CORS. Також було активовано middleware для автоматичного розбору JSON-даних у запитах.

Однією з центральних функцій серверу є обробка форми зворотного зв’язку, розміщеної на клієнтській частині сайту. Ця форма надсилає POST-запит на маршрут /contact, який приймає вхідні дані: ім’я, email, номер телефону, повідомлення, ознаку згоди з політикою конфіденційності та токен reCAPTCHA. Усі отримані дані обов’язково проходять валідацію згідно зі схемою, визначеною у файлі validation.js (див. рис. 3.17).

```
const yup = require('yup')

const contactSchema = yup.object().shape({
  name: yup.string().required(),
  email: yup.string().email().required(),
  phone: yup
    .string()
    .matches(/^[\d\s()+-]+$/, 'Invalid phone')
    .required(),
  message: yup.string().max(1000),
  policy: yup.boolean().oneOf([true]),
  recaptcha: yup.string().required()
})

module.exports = contactSchema
```

Рисунок 3.17 – Файл validation.js зі схемою перевірки полів форми за допомогою Yup

Бібліотека Yup дозволяє задавати суворі правила до кожного поля. Наприклад, email має відповідати шаблону електронної адреси, номер телефону – певному регулярному виразу, поля з текстом не можуть бути порожніми, а токен reCAPTCHA є обов’язковим для підтвердження, що запит було надіслано людиною, а не ботом.

У разі проходження валідації, сервер формує SQL-запит, який вставляє отримані дані у відповідну таблицю contacts у базі даних PostgreSQL. Для підключення до БД використовується модуль pg, а всі параметри підключення зберігаються у змінних середовища .env, що дозволяє безпечно управляти конфігурацією. З’єднання з базою реалізоване у окремому модулі db.js, що експортує пул з’єднань для повторного використання в інших модулях (див. рис. 3.18).

```
CREATE TABLE contacts (
  id SERIAL PRIMARY KEY,
  name TEXT NOT NULL,
  email TEXT NOT NULL,
  phone TEXT,
  message TEXT,
  policy BOOLEAN NOT NULL,
  recaptcha TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE DATABASE lukashevarchitects;
DROP TABLE contacts;

SELECT *
FROM contacts;
```

Рисунок 3.18 – SQL-запит для вставки даних у таблицю contacts

У випадку успішного збереження даних, сервер повертає клієнту відповідь зі статусом 201 та повідомленням про успіх. Якщо ж відбувається помилка на етапі валідації, сервер відповідає повідомленням з описом порушення та статусом 400. Усі інші помилки, наприклад, пов’язані з базою даних, обробляються як внутрішні помилки серверу зі статусом 500.

Таким чином, обробка запиту виглядає повністю асинхронною, передбачає всі основні сценарії (успішне завершення, помилки користувача, внутрішні збої) і є достатньо гнучкою для розширення або доповнення. Усе логування винесено в консоль, і на етапі розробки повідомлення про запуск серверу дозволяють легко перевірити, що він активний і працює на очікуваному порту (див. рис. 3.19).

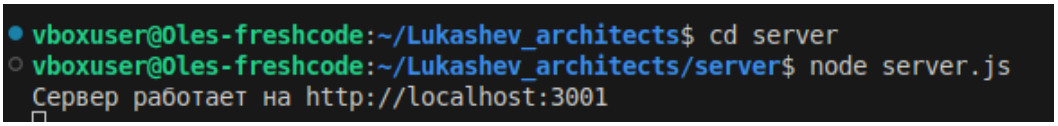


Рисунок 3.19 – Повідомлення у консолі при запуску Node.js-сервера

Завдяки модульному підходу до організації коду, сервер легко підтримувати та масштабувати. Окремо винесені частини логіки (валидація, робота з БД) дозволяють уникати дублювання коду та підвищити читабельність. Вся система готова до інтеграції з іншими сервісами, наприклад, поштовими API, системами авторизації або аналітики.

3.3 Тестування вебзастосунку

Тестування вебзастосунку архітектурної майстерні є важливим етапом розробки, оскільки воно дозволяє виявити та усунути потенційні помилки в роботі інтерфейсу, адаптивності, інтерактивних елементів і продуктивності ще до запуску проєкту у відкритий доступ.

Одним із перших аспектів тестування стала перевірка коректності клієнтської валідації даних у формі зворотного зв’язку. Саме ця форма є

головним засобом комунікації між потенційними клієнтами та архітекторами, тому вона повинна працювати надійно й інтуїтивно. Валідація була реалізована за допомогою бібліотек Formik і Yup, які забезпечують перевірку введених користувачем даних ще до надсилання запиту на сервер.

Було проведено ряд ручних тестів для перевірки реакції форми на різні сценарії неправильного введення даних. Зокрема, тестувалася обробка випадків, коли користувач:

– не заповнив обов'язкові поля (ім'я, електронну пошту, повідомлення);
– ввів некоректну email-адресу (без символу «@»);
– написав надто коротке повідомлення (менше 10 символів).

У кожному з цих випадків інтерфейс надавав відповідні повідомлення про помилку, які з'являлися після втрати фокусу з поля або під час спроби надіслати форму (рис. 3.20). Це дозволяє користувачеві швидко виявити й виправити помилки без втрати часу.

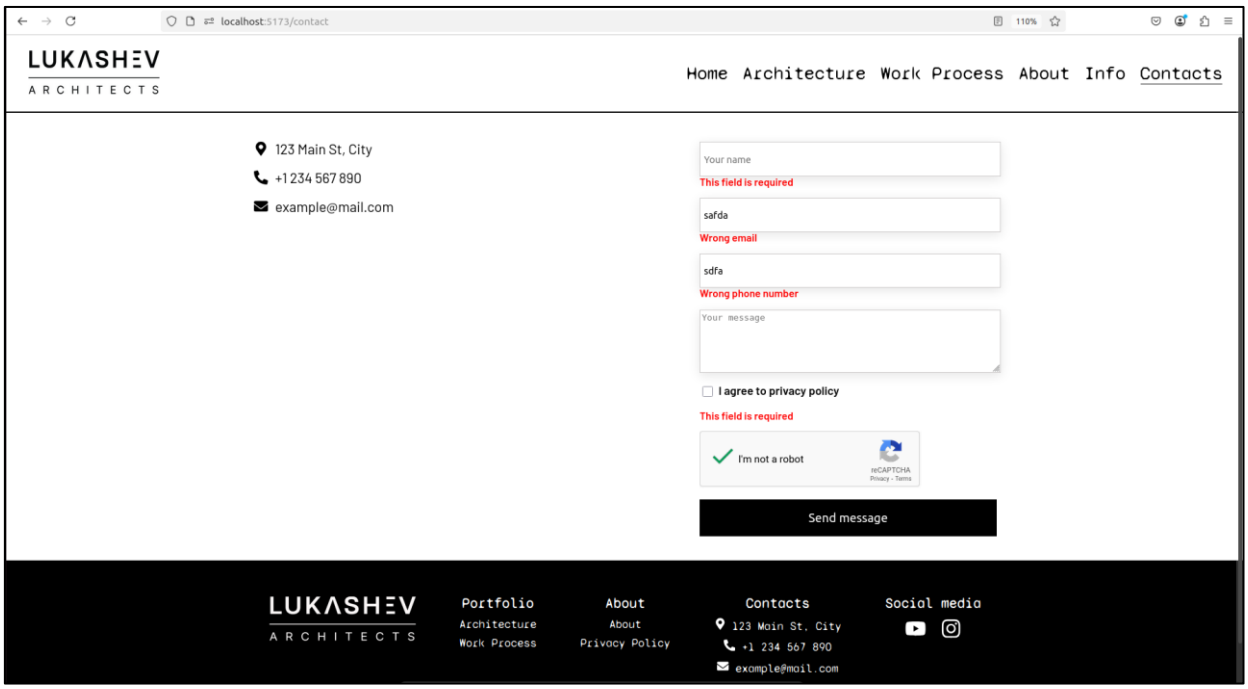


Рисунок 3.20 – Валідація даних у формі зворотного зв'язку

У разі правильного заповнення всіх полів форма дозволяє відправлення повідомлення, демонструючи повідомлення про успішне надсилання. Такий

підхід до валідації гарантує базову перевірку введених даних і мінімізує кількість некоректних або порожніх запитів із боку користувачів.

Оскільки сайт виконує функцію онлайн-візитки, особливу увагу було приділено стабільності роботи контактної форми, правильному відображенню портфоліо проєктів та коректній роботі навігації. Була проведена серія тестів, що перевіряли як валідацію даних користувача (як описано вище), так і обробку повідомлень при успішному або неуспішному надсиланні. Для захисту від спаму також було протестовано роботу інтегрованої Google ReCAPTCHA, яка не дозволяє ботам автоматично надсилати повідомлення через форму. У випадку успішного заповнення форми система генерує підтвердження для користувача, а також журналює подію для адміністратора. При виникненні помилок система реагує відповідним повідомленням, не порушуючи загальний користувацький досвід (рис. 3.21).

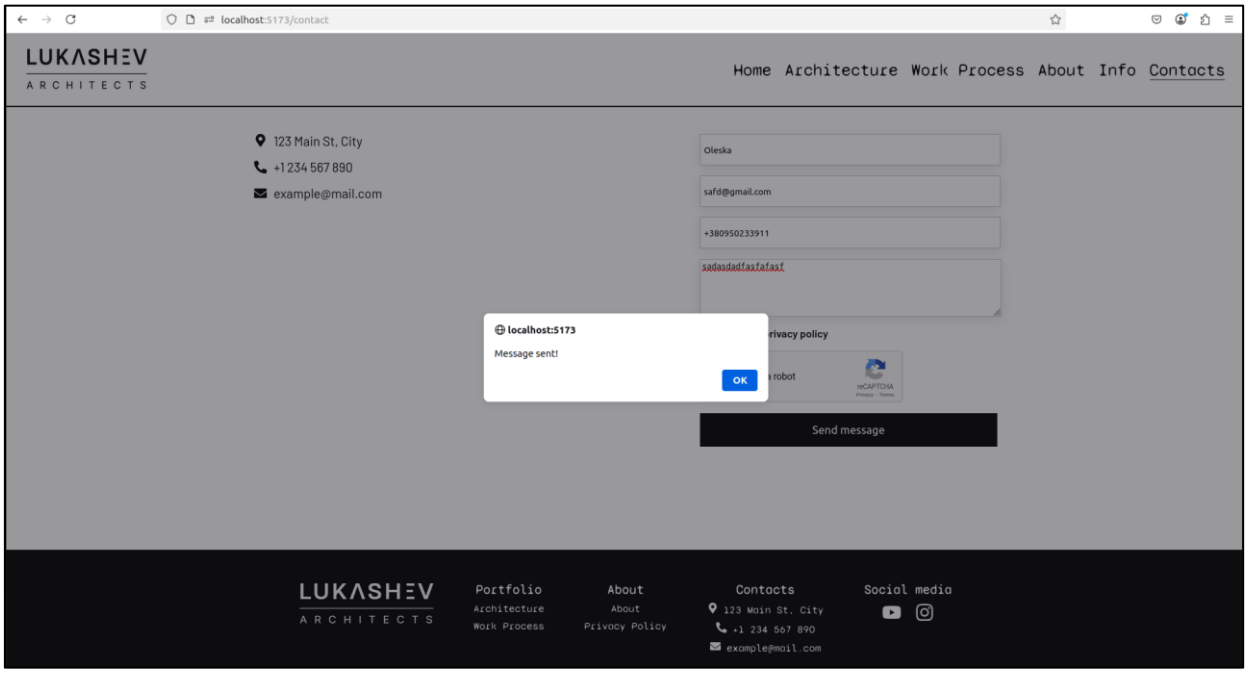


Рисунок 3.21 – Тестування форми зворотного зв’язку із повідомленням про помилку/успіх

Окрему увагу було приділено адаптивності інтерфейсу. Тестування проводилося на різних пристроях, зокрема на настільних ПК, планшетах і мобільних телефонах. Перевірялося, як саме змінюється розміщення елементів

сайту при різних розмірах екрану, зокрема – перетворення горизонтального меню в мобільне «бургер-меню», поведінка галереї зображень портфоліо, а також збереження читабельності текстів і логіки прокрутки. Адаптивне тестування дало змогу переконатися, що сайт залишається функціональним і зручним незалежно від розміру екрана чи типу пристрою. Всі зображення проєктів коректно масштабуються, не обрізаються та зберігають високу якість, використовуючи техніки оптимізації, такі як lazy loading і WebP-формати (див. рис. 3.22).



Рисунок 3.22 – Тест адаптивності на мобільному пристрої та планшеті

Також було протестовано роботу внутрішньої навігації між сторінками – усі посилання та переходи працюють безперебійно завдяки використанню React Router. Натискання на пункти меню викликає зміну сторінки без перезавантаження. Активні пункти підсвічуються, що покращує зручність користування. У разі введення некоректної адреси в URL користувач потрапляє на сторінку з повідомленням про помилку 404, що свідчить про передбачення нестандартних сценаріїв використання.

Одним із елементів функціонального тестування стало тестування інтерактивності елементів футера, зокрема кнопок контактів і посилань на соціальні мережі. Оскільки футер сайту містить ключову інформацію – адресу офісу, телефон, електронну пошту та посилання на соціальні платформи – важливо було впевнитися, що всі ці елементи працюють очікувано та зручно для користувача.

Для тестування було перевірено, що натискання на іконку телефону викликає додаток для дзвінків на мобільних пристроях, натискання на електронну пошту відкриває поштовий клієнт із правильно підставленою адресою отримувача.

Іконки соціальних мереж перенаправляють на відповідні сторінки у новій вкладці, не порушуючи навігацію на сайті. Особливу увагу було приділено відповідності всіх посилань правильним URL, а також наявності атрибутів rel="noopener noreferrer" для забезпечення безпеки при відкритті зовнішніх сторінок у нових вкладках.

Крім того, було проведено тестування навігації за допомогою клавіатури та перевірено, що всі інтерактивні елементи мають чіткий фокус при переміщенні через клавішу Tab, що є важливим з точки зору доступності. Усі кнопки та іконки мали підказки (tooltips) або доступні альтернативні тексти для користувачів, які використовують скрінрідери.

Таке тестування дозволяє гарантувати, що користувачі зможуть безперешкодно зв’язатися з архітектурною майстернею або знайти її профілі у соцмережах, що сприяє налагодженню комунікації та підвищенню довіри до компанії (див. рис. 3.23).

Таким чином, після ретельного комплексного тестування вебзастосунок архітектурної майстерні продемонстрував високу стабільність, якість адаптації під різні пристрої та зручність у користуванні. Завдяки тестуванню вдалося забезпечити надійність усіх ключових елементів, що сприяє позитивному сприйняттю сайту відвідувачами та підвищенню довіри до бренду майстерні.

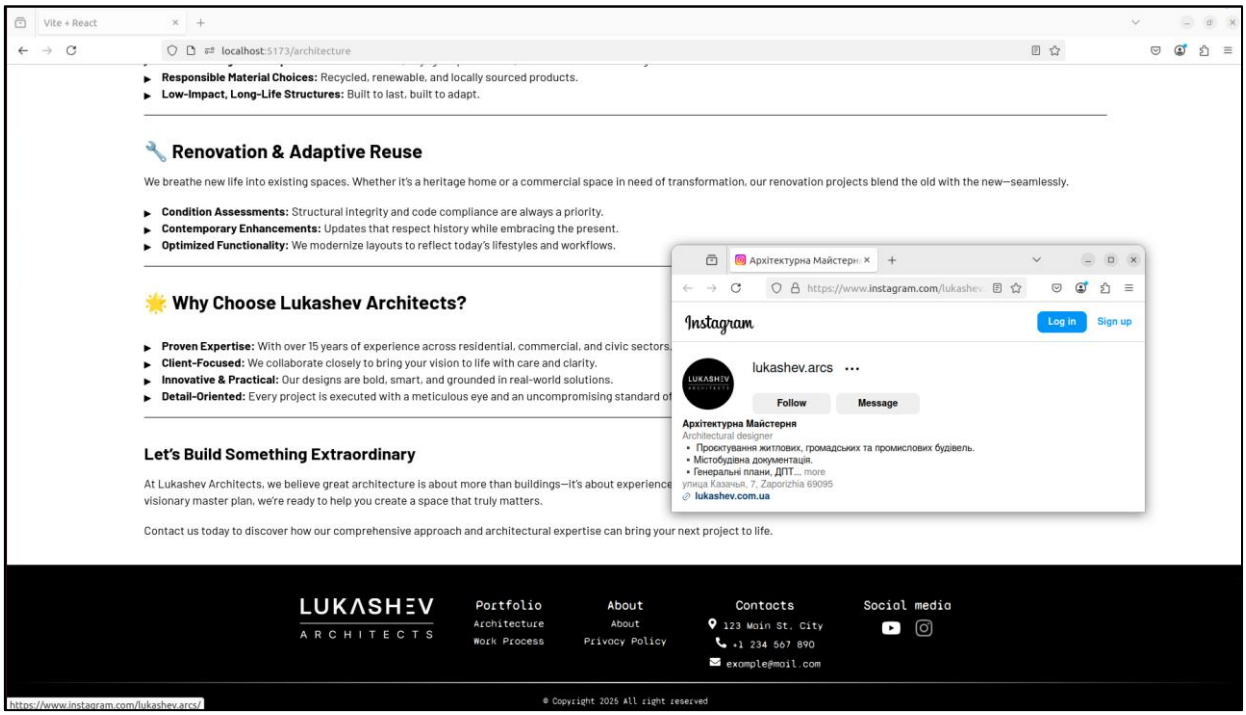


Рисунок 3.23 – Тестування функціональності кнопок контактів та соцмереж у футері вебзастосунку

3.4 Результати реалізації та тестування

Підсумовуючи реалізацію та тестування розробленого вебзастосунку, можна впевнено стверджувати, що створений сайт-візитка архітектурної майстерні відповідає поставленим цілям та технічним вимогам. Основний акцент у процесі розробки було зроблено на візуальну привабливість, простоту навігації та доступність основної інформації про майстерню.

Інтерфейс вебзастосунку був реалізований з урахуванням сучасних принципів UI/UX-дизайну. Користувачі мають змогу швидко ознайомитись з напрямками діяльності майстерні, прикладами реалізованих проєктів, отримати контактну інформацію та перейти до соціальних мереж. Сайт структурується логічно – починаючи зі знайомства з брендом на головній сторінці, і завершуючи можливістю негайного зв’язку через форму. Така ієрархія дозволяє формувати чітке враження про діяльність майстерні буквально за кілька кліків.

Всі секції сайту адаптовані для перегляду з мобільних пристроїв, що забезпечує зручність користування незалежно від типу екрану. Проведено тестування на найбільш поширених типах пристроїв (мобільні телефони з діагоналлю 5-7 дюймів, планшети, ноутбуки та монітори з роздільністю Full HD і вище). Результати засвідчили, що жоден із функціональних елементів не виходить за межі екрана, а взаємодія з сайтом залишається інтуїтивною. Адаптивна верстка з використанням медіа-запитів забезпечила правильне масштабування зображень та збереження загальної композиції інтерфейсу.

Технічна реалізація включала інтеграцію багатосторінкової структури за допомогою React, реалізацію анімаційних ефектів для підвищення привабливості, та валідацію контактної форми з використанням Formik та Yup. Крім того, впроваджено функціональність переходу між сторінками без перезавантаження, що позитивно впливає на швидкодію вебзастосунку.

Тестування вебзастосунку охопило перевірку працездатності всіх інтерактивних елементів, включаючи кнопки навігації, соціальні посилання у футері, форму зворотного зв’язку та відкриття галереї проєктів. Для кожного з інтерактивних блоків було складено перелік очікуваних сценаріїв поведінки (expected outcomes), після чого вручну перевірено відповідність фактичної поведінки інтерфейсу. Наприклад, при натисканні на зображення проєкту очікується відкриття збільшеної версії з можливістю перегортання; така функція була реалізована і пройшла успішне тестування на різних пристроях.

Окрему увагу було приділено перевірці адаптивності – сайт коректно відображається на різних пристроях і в усіх сучасних браузерах. В рамках тестування доступності перевірено фокусування елементів при навігації з клавіатури, наявність альтернативних текстів до зображень та правильну семантичну структуру HTML-коду. Це особливо важливо в контексті інклюзивності вебсередовища, оскільки дає змогу взаємодіяти із сайтом не лише користувачам із повноцінним зором та моторикою, а й тим, хто використовує скрінрідери або альтернативні методи введення. Семантична структура також позитивно впливає на SEO, підвищуючи шанси ресурсу з’являтись у релевантних пошукових запитах.

Окремо було перевірено час завантаження основних сторінок при використанні інструментів браузера та симуляції повільного інтернет-з’єднання. Завдяки оптимізації зображень (lazy loading, WebP), використанню кешування та модульній структурі компонентів React, сайт демонструє швидке завантаження – менш як 2 секунди при типовому 4G-з’єднанні.

Результати тестування свідчать про високу стабільність роботи вебзастосунку, відсутність критичних помилок і швидке завантаження сторінок. Усі функціональні елементи працюють відповідно до очікувань, що підтверджує завершеність і готовність сайту до публічного використання.

Таким чином, можна говорити про успішне поєднання сучасних технологій, дизайнерських рішень і практик тестування, що в результаті дало змогу створити не просто візитку, а повноцінний інструмент презентації бренду архітектурної майстерні в цифровому просторі.

Отже, створений вебзастосунок повністю відповідає сучасним стандартам як з технічної, так і з естетичної точки зору, надаючи архітектурній майстерні ефективний інструмент для презентації своїх робіт і комунікації з клієнтами.

ВИСНОВКИ

У результаті виконання проєкту було створено вебзастосунок – сайт-візитку архітектурної майстерні, що забезпечує ефективну презентацію діяльності студії, демонстрацію реалізованих проєктів та зручну комунікацію з потенційними клієнтами. В процесі роботи проаналізовано цільову аудиторію, сформульовано вимоги, обрано оптимальні інструменти й реалізовано інтерфейс відповідно до сучасних стандартів веброзробки.

Клієнтська частина побудована з використанням бібліотеки React, що забезпечило компонентну структуру, зручну маршрутизацію та інтерактивність. Сучасний дизайн досягнуто завдяки SCSS, адаптивній верстці та анімаційним ефектам, що покращують UX.

Реалізовано контактну форму з валідацією введених даних (Formik, Yup) та Google ReCAPTCHA для захисту від спаму. Структура сайту включає головну сторінку, портфоліо, опис послуг, контакти та посилання на соціальні мережі.

Проведене тестування підтвердило стабільність роботи, коректність усіх функцій та повну адаптивність до мобільних пристроїв і популярних браузерів. Особливо перевірено роботу меню, форми зворотного зв’язку та зовнішніх посилань.

У майбутньому можливе розширення функціоналу: мультимовність, адміністрування вмісту та додавання нових проєктів. Отримані результати підтверджують успішність реалізації задач і практичну цінність вебзастосунку для цифрової присутності архітектурної студії.

ПЕРЕЛІК ПОСИЛАНЬ

1. Практичні поради з верстки. *CSS-Tricks*. URL: https://css-tricks.com/ (дата звернення: 04.03.2024).
2. Документація. *Express.js*. URL: https://expressjs.com/ (дата звернення: 14.03.2025).
3. Огляд. *Formik Documentation*. URL: https://formik.org/docs/overview (дата звернення: 24.03.2024).
4. Документація v3. *Google ReCAPTCHA*. URL: https://developers.google.com/recaptcha/docs/v3 (дата звернення: 30.03.2024).
5. HTML, CSS, JavaScript. *MDN Web Docs*. URL: https://developer.mozilla.org/ (дата звернення: 18.03.2024).
6. Документація. *Node.js*. URL: https://nodejs.org/en/docs (дата звернення: 14.03.2025).
7. Learn. *Node.js*. URL: https://nodejs.dev/en/learn (дата звернення: 14.03.2025).
8. Документація. *PostgreSQL*. URL: https://www.postgresql.org/docs/ (дата звернення: 14.03.2025).
9. Learn. *React*. URL: https://react.dev/learn (дата звернення: 19.04.2024).
10. Your first component. *React*. URL: https://react.dev/learn/your-first-component (дата звернення: 14.03.2025).
11. Passing props to a component. *React*. URL: https://react.dev/learn/passing-props-to-a-component (дата звернення: 14.03.2025).
12. State: a component’s memory. *React*. URL: https://react.dev/learn/state-a-components-memory (дата звернення: 14.03.2025).
13. Документація. *React Router*. URL: https://reactrouter.com/en/main (дата звернення: 22.03.2024).

14. Інструкція. *SCSS Guide*. URL: https://sass-lang.com/guide/ (дата звернення: 20.03.2024).

15. Огляд пакета. *Yup (npm)*. URL: https://www.npmjs.com/package/yup (дата звернення: 26.03.2024).

16. Wieruch R. The Road to React: Your journey to master plain yet pragmatic React.js. Self-published, 2021. 250 p.

17. Wolf J. HTML & CSS: The Comprehensive Guide to Excelling in HTML5 and CSS3 for Responsive Web Design, Dynamic Content, and Modern Layouts. Rheinwerk Computing, 2020.1040 p.

18. Minnick C. JavaScript All-in-One For Dummies. John Wiley & Sons, 2023. 928 p.

19. Gascón U. Node.js for Beginners: A comprehensive guide to building efficient, full-featured web applications with Node.js. Packt Publishing, 2023. 296 p.

20. Pirozzi E. Learn PostgreSQL: Use, manage, and build secure and scalable databases with PostgreSQL 16. Packt Publishing, 2023.446 p.