МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ВЕБЗАСТОСУНКУ ПРАЦЕВЛАШТУВАННЯ З ВИКОРИСТАННЯМ MERN СТЕКУ»**

Виконав: студент 2 курсу, групи 8.1213-з
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Д.О. Салякін
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Кудін О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент професор кафедри комп'ютерних наук, професор, д.т.н. Шило Г.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

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

Салякіну Дмитру Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебзастосунку працевлаштування з використанням MERN стеку

керівник роботи Кудін Олексій Володимирович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 384-с

2. Строк подання студентом роботи 25.11.2024 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. Розробка та тестування вебзастосунку.

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 07.03.2024 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.	15.04.2024	
2.	Збір вихідних даних.	27.05.2024	
3.	Обробка методичних та теоретичних	27.06.2024	
	джерел.		
4.	Розробка першого та другого розділу.	30.08.2024	
5.	Розробка третього розділу.	31.10.2024	
6.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	14.12.2024	

Студент _____ Д.О. Салякін
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.В. Кудін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка вебзастосунку працевлаштування з використанням MERN стеку»: 59 с., 21 рис., 1 табл., 20 джерел.

ВЕБАРХІТЕКТУРА, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, МАСШТАБОВАНІСТЬ, ПІДБІР ПЕРСОНАЛУ, ПОШУК РОБОТИ, EXPRESS.JS, MERN СТЕК, MONGODB, NODE.JS, NOSQL, REACT, RESTFUL API.

Об’єкт дослідження – методи та засоби автоматизації процесів пошуку роботи та підбору персоналу з використанням сучасних вебтехнологій, зокрема MERN стеку.

Мета роботи: дослідження та реалізація методів створення масштабованого вебзастосунку для автоматизації процесів пошуку роботи та підбору персоналу на основі MERN стеку.

Предмет дослідження – технології та архітектурні рішення для розробки вебзастосунку з використанням MERN стеку.

Методи дослідження: системний аналіз технічної літератури, технічний аналіз вимог, архітектурне прототипування, програмна інженерія, функціональне тестування.

У роботі досліджено можливості MERN стеку для розробки масштабованих вебзастосунків. Розроблено архітектуру системи з використанням MongoDB для гнучкої роботи з даними. Реалізовано серверну частину на Node.js та Express.js, створено клієнтський інтерфейс на React. Результатом є вебзастосунок для пошуку роботи та підбору персоналу, адаптований до специфіки ринку праці в Україні.

SUMMARY

Master's qualifying paper "Development of an Employment Web Application Using MERN Stack": 59 pages, 21 figures, 1 table, 20 references.

MERN STACK, MONGODB, EXPRESS.JS, REACT, NODE.JS, RESTFUL API, WEB ARCHITECTURE, SCALABILITY, NOSQL, CLIENT-SERVER ARCHITECTURE, JOB SEARCH, RECRUITMENT.

The object of study is methods and means of automating job search and recruitment processes using modern web technologies, particularly the MERN stack.

The aim of the study is to research and implement methods and tools for developing a scalable web application for automating job search and recruitment processes based on the MERN stack.

The subject of research is technologies and architectural solutions for web application development using the MERN stack, including distributed systems design, performance optimization, and user interface development.

Research methods: systematic analysis of technical literature, technical requirements analysis, architectural prototyping, software engineering, functional testing, performance analysis.

The paper investigates the capabilities of the MERN stack for developing scalable web applications. The choice of MongoDB as a database is justified, providing flexible data schema and efficient handling of various information types. The system architecture has been developed to ensure effective data organization and processing. The server-side implementation using Node.js and Express.js provides reliable database interaction. The client-side, created using React, delivers a modern adaptive interface. The result is a web application that simplifies the job search and recruitment process, adapted to the specifics of the Ukrainian labor market.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Аналіз предметної області..... 10
1.1 Огляд передумов та сучасного контексту 10
1.2 Технічний огляд стеку та аналіз поширених стеків 11
2 Проєктування та архітектура вебзастосунку..... 14
2.1 Загальний опис архітектури системи..... 14
2.1.1 Основні компоненти системи 14
2.1.2 Взаємодія компонентів..... 15
2.1.3 Ролі користувачів системи 16
2.2 Моделювання поведінки системи 17
2.2.1 Діаграма прецедентів..... 17
2.2.2 Діаграма послідовності створення вакансії 18
2.3 Прототипування та візуалізація користувацького інтерфейсу..... 19
2.3.1 Визначення центральних сутностей системи 21
2.3.2 Створення макетів основних сторінок..... 22
2.3.3 Візуалізація елементів інтерфейсу 23
3 Розробка, тестування та розгортання додатка..... 25
3.1 Розробка клієнтської частини (Frontend)..... 25
3.1.1 Архітектурний підхід 26
3.1.2 Управління станом..... 27
3.1.3 Компонентна структура 28
3.1.4 Взаємодія з API 29
3.2 Розробка серверної частини (Backend) 30
3.2.1 Архітектура API сервера 31

3.2.2 Структура серверної частини 32
3.2.3 Моделі даних 33
3.2.4 Реалізація логіки фільтрації даних 34
3.2.5 Тестування API 36
3.2.6 Інтеграція фронтенду та бекенду 37
3.3 Проєктування бази даних 38
3.3.1 Структура колекцій 39
3.3.2 Початкове наповнення бази даних 40
3.3.3 Оптимізація та індексація 42
3.4 Структура вебзастосунку 43
3.4.1 Клієнтська частина та її файлова структура 43
3.4.2 Логіка серверної частини 46
3.5 Етапи розробки вебзастосунку 48
3.5.1 Використані бібліотеки та утиліти 49
3.5.2 Створення базового каркасу проєкту 49
3.5.3 Розширення функціоналу 50
3.5.4 Тестування та відлагодження 51
3.5.5 Розгортання застосунку 53
Висновки 56
Перелік посилань 58

ВСТУП

В умовах стрімкої цифровізації та трансформації ринку праці України особливої актуальності набуває створення ефективних інструментів для автоматизації процесів працевлаштування. Сучасні вебтехнології, зокрема MERN стек (MongoDB, Express.js, React, Node.js), відкривають нові можливості для розробки масштабованих платформ, здатних забезпечити ефективну взаємодію між роботодавцями та шукачами роботи з урахуванням регіональних особливостей.

Розвиток вебплатформ працевлаштування пройшов шлях від простих дошок оголошень до складних систем з розширеними можливостями пошуку та підбору персоналу. Проте існуючі рішення часто не враховують специфіку локальних ринків праці та потреби територіальних громад, що особливо актуально в умовах децентралізації.

Об’єктом дослідження є методи та засоби автоматизації процесів пошуку роботи та підбору персоналу з використанням сучасних вебтехнологій, зокрема MERN стеку.

Предметом дослідження виступають технології та архітектурні рішення для розробки вебзастосунку з використанням MERN стеку, включаючи проєктування розподілених систем, оптимізацію продуктивності та розробку користувацьких інтерфейсів.

Метою роботи є дослідження та реалізація методів створення масштабованого вебзастосунку для автоматизації процесів пошуку роботи та підбору персоналу на основі MERN стеку. Для досягнення мети визначено наступні завдання:

- провести аналіз технологій MERN стеку та їх застосування у веброзробці;
- дослідити архітектурні паттерни для проєктування масштабованих систем;

- розробити архітектуру вебзастосунку з урахуванням регіональних особливостей;
- реалізувати серверну частину з використанням Node.js та Express.js;
- створити клієнтську частину на базі React;
- провести тестування та оптимізацію системи.

Методологічною основою дослідження є системний підхід до аналізу та розробки програмного забезпечення. У роботі використано методи системного аналізу технічної літератури, технічного аналізу вимог, архітектурного прототипування, програмної інженерії та функціонального тестування.

Наукова новизна роботи полягає в розробці та обґрунтуванні архітектурних рішень для створення вебзастосунку на базі MERN стеку, що забезпечують гнучкість структури даних та ефективність пошуку з урахуванням регіональної специфіки ринку праці.

Практичне значення отриманих результатів підтверджується створенням працюючого вебзастосунку з гнучкою архітектурою, що враховує особливості регіональних ринків праці та забезпечує можливість подальшого масштабування відповідно до зростаючих потреб територіальних громад.

Робота складається зі вступу, трьох розділів, висновків та списку використаних джерел. У першому розділі проведено аналіз предметної області та технологій MERN стеку. Другий розділ присвячено проєктуванню та архітектурі вебзастосунку. У третьому розділі описано процес розробки, тестування та розгортання системи. Загальний обсяг роботи становить 59 сторінок, включаючи 21 рисунок та 1 таблицю. Список використаних джерел містить 20 найменувань.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд передумов та сучасного контексту

Розвиток сучасних вебтехнологій відкриває нові можливості для створення спеціалізованих платформ працевлаштування, орієнтованих на потреби локальних ринків праці. В умовах децентралізації та розвитку територіальних громад в Україні зростає потреба в інструментах, що забезпечують ефективну взаємодію між роботодавцями та шукачами роботи на регіональному рівні. Особливої актуальності це питання набуває з огляду на суттєву географічну нерівномірність можливостей працевлаштування між великими містами та малими населеними пунктами.

Аналіз статистичних даних Державної служби зайнятості свідчить про значну диспропорцію у представленості вакансій між обласними центрами та територіальними громадами. В умовах воєнного стану та пов’язаної з ним внутрішньої міграції населення, ця ситуація додатково загострюється, що підкреслює важливість розробки ефективних інструментів локального пошуку роботи та підбору персоналу [1].

Використання сучасного стеку технологій MERN (MongoDB [12], Express.js [5], React [16], Node.js [13]) створює технічну базу для вирішення цих викликів. Особливості MongoDB у роботі з геопросторовими даними забезпечують можливість точної геолокації вакансій, а гнучкість React дозволяє створити зручний користувацький інтерфейс з урахуванням специфічних потреб локальних спільнот. Така технічна база дозволяє реалізувати ряд важливих функціональних можливостей:

- пошук роботи в межах конкретної територіальної громади;
- визначення доступних вакансій у заданому радіусі від місця проживання;
- оцінка транспортної доступності місця роботи;
- комбінування локального пошуку з можливостями віддаленої роботи.

Обраний технологічний стек забезпечує необхідну гнучкість для поетапного впровадження системи, починаючи з окремих регіонів, з можливістю подальшого масштабування. Архітектурні особливості MERN стеку дозволяють ефективно працювати з географічно розподіленою інформацією та адаптувати систему до зростаючих потреб користувачів.

Соціально-економічна значущість такого підходу полягає у:

- підтримці розвитку локальних ринків праці;
- сприянні економічному розвитку територіальних громад;
- зменшенні трудової міграції з малих населених пунктів;
- створенні рівних можливостей працевлаштування незалежно від місця проживання.

Таким чином, розробка вебзастосунку для пошуку роботи з акцентом на регіональні особливості є актуальним завданням, що поєднує технічні інновації з важливими соціально-економічними цілями та відповідає сучасним тенденціям децентралізації в Україні.

1.2 Технічний огляд стеку та аналіз поширених стеків

Створення вебзастосунків є багатогранним процесом, що вимагає ретельного вибору технологій для забезпечення якості, продуктивності та масштабованості. Вебстек – це комплекс інструментів і технологій, які забезпечують роботу серверної частини, взаємодію з базою даних і розробку інтерактивного інтерфейсу користувача. Вибір конкретного стеку обумовлюється багатьма факторами, такими як вимоги проєкту, рівень підготовки команди, доступний бюджет і перспективи масштабування. Саме цей вибір визначає не тільки технічні можливості застосунку, але й ефективність його подальшої підтримки та адаптації до змін.

Під час аналізу технологічних стеків [4, 19], я детально розглянув кілька популярних рішень, зокрема LAMP, MEAN, MERN, MEVN, Django та Ruby on

Rails. Кожен з них має свої переваги та недоліки, і вибір залежить від конкретних завдань і контексту використання (див. табл. 1.1).

Таблиця 1.1 – Порівняльна таблиця технологічних стеків

Характеристика	**LAMP**	**MEAN**	**MERN**	**MEVN**	**Django**
Масштабованість	Середня	Висока	Висока	Висока	Середня
Продуктивність	Середня	Висока	Висока	Висока	Висока
Гнучкість схеми даних	Низька	Висока	Висока	Висока	Низька
Екосистема	Дуже велика	Велика	Дуже велика	Середня	Велика
Поріг входження	Низький	Високий	Середній	Середній	Середній
Спільнота розробників	Дуже велика	Велика	Дуже велика	Середня	Велика

LAMP (Linux, Apache, MySQL, PHP/Python/Perl) є класичним вибором для багатьох проєктів завдяки своїй стабільності, зрілості та широкій підтримці [19]. Проте його обмежена масштабованість і менш висока продуктивність у порівнянні з сучасними стековими рішеннями обмежують його застосування в масштабних проєктах.

MEAN (MongoDB, Express.js, Angular, Node.js) вирізняється своєю продуктивністю та використанням єдиної мови JavaScript для всіх етапів розробки [4]. Однак його складність налаштування та відносно менша кількість готових рішень можуть викликати труднощі в команд, що мають обмежений досвід роботи з цими технологіями.

MERN (MongoDB, Express.js, React, Node.js), який я обрав для свого проєкту, базується на ідеях MEAN, але використовує React замість Angular. Ця заміна дозволяє створювати більш гнучкі та інтерактивні інтерфейси завдяки широким можливостям React. Також популярність React сприяє доступності численних бібліотек і спрощує розробку.

MEVN (MongoDB, Express.js, Vue.js, Node.js) є схожим на MERN, але пропонує використання Vue.js, відомого своєю легкістю і зручністю. Django та Ruby on Rails, у свою чергу, є відмінними варіантами для швидкої розробки, але їхня гнучкість в інтерактивних інтерфейсах поступається стекам, побудованим на основі JavaScript.

Чому MERN? Ретельно зваживши всі фактори, я зупинився на MERN для реалізації вебзастосунку з пошуку роботи та підбору персоналу. Використання єдиного мовного стеку на базі JavaScript значно спрощує розробку і підтримку проєкту. Node.js забезпечує високу продуктивність завдяки асинхронній обробці запитів, а MongoDB дозволяє працювати з великими обсягами даних. React, своєю чергою, забезпечує зручність розробки інтерактивних інтерфейсів, що є критично важливим для користувачів.

Роль кожної технології в проєкті. MongoDB виступає основною базою даних для зберігання інформації про користувачів, вакансії та резюме. Її гнучкість у роботі зі схемами дозволяє легко адаптуватися до змін і масштабувати систему. Express.js забезпечує створення серверної частини, маршрутизацію запитів і взаємодію з MongoDB. React.js використовується для розробки клієнтського інтерфейсу, включаючи форми для створення та перегляду вакансій. Node.js виконує роль серверного середовища для обробки запитів та інтеграції всіх компонентів.

Висновок. MERN-стек виявився ідеальним вибором для створення сучасного, продуктивного та масштабованого вебзастосунку. Цей вибір забезпечує швидкість розробки, високу продуктивність системи та гнучкість у впровадженні нових функцій, що є критично важливим у динамічному середовищі ринку праці.

2 ПРОЄКТУВАННЯ ТА АРХІТЕКТУРА ВЕБЗАСТОСУНКУ

Від якості проведеного процесу проєктування та архітектури будь-якого вебзастосунку залежить надійність та ефективність роботи системи, а також можливості його масштабування і розширення функціоналу в майбутньому.

2.1 Загальний опис архітектури системи

Вебзастосунок з пошуку роботи розроблено за принципами клієнт-серверної архітектури з використанням сучасного стеку технологій MERN (MongoDB [12], Express.js [5], React [16], Node.js [13]).

2.1.1 Основні компоненти системи

Основні компоненти системи, які забезпечують її функціональність, складаються з трьох ключових частин: клієнтської, серверної та бази даних. Кожен з цих елементів виконує свою унікальну роль, що дозволяє досягти цілісності та ефективності роботи вебзастосунку.

Клієнтська частина була реалізована за допомогою React [16] у поєднанні з TypeScript [10]. Вона відповідає за створення користувацького інтерфейсу та забезпечує інтерактивність взаємодії з системою. React дозволяє легко керувати станом додатку, а використання TypeScript підвищує надійність та передбачуваність роботи завдяки статичній типізації. У цьому компоненті користувачі мають змогу виконувати такі дії, як перегляд вакансій, заповнення резюме або пошук релевантних пропозицій.

Серверна частина побудована на основі Node.js [13] із використанням Express.js [5], що забезпечує високопродуктивну обробку запитів і управління бізнес-логікою. Сервер виступає як посередник між клієнтом та базою даних,

відповідаючи за обробку запитів API, реалізацію механізмів авторизації, а також захист даних. Завдяки використанню Express.js вдалося створити гнучку архітектуру, яка легко адаптується до змін і розширення функціональності.

База даних, розгорнута на платформі MongoDB Atlas [12], виступає основним сховищем даних. Вона зберігає інформацію про користувачів, вакансії та резюме, забезпечуючи швидкий доступ до даних навіть за значних обсягів інформації. Використання MongoDB Atlas як хмарного рішення дозволяє забезпечити високу доступність, масштабованість та зручність адміністрування, що є важливим для підтримки стабільної роботи системи.

2.1.2 Взаємодія компонентів

Взаємодія між основними компонентами системи організована за чіткою послідовністю, що забезпечує ефективний обмін даними та узгодженість роботи всіх складових.

На початковому етапі клієнтська частина, реалізована на React, формує і надсилає запити до серверної частини. Це здійснюється через REST API, що забезпечує стандартизований формат передачі даних між фронтендом та бекендом.

Серверна частина, побудована на основі Node.js із використанням Express.js, приймає запити від клієнта. Вона аналізує отримані дані, виконує необхідну обробку та, у разі потреби, звертається до бази даних для отримання або оновлення інформації. MongoDB Atlas, як сховище даних, забезпечує швидкий доступ до потрібних записів, що є ключовим етапом обробки запитів.

Після завершення обробки сервер повертає результати клієнтській частині. Це можуть бути дані про вакансії, підтвердження виконаних операцій чи повідомлення про помилки. Отримавши відповідь, клієнтська частина оновлює інтерфейс користувача відповідно до змісту отриманих даних. Наприклад, це може включати відображення списку вакансій, оновлення деталей резюме або інші динамічні зміни у візуальній частині застосунку.

Такий підхід до організації взаємодії компонентів дозволяє забезпечити узгодженість дій та підтримувати інтерактивність роботи системи.

2.1.3 Ролі користувачів системи

У системі реалізовано підтримку чотирьох основних ролей користувачів, кожна з яких має свій набір функцій і доступ до певного функціоналу.

Неавторизований користувач має обмежені можливості, що дозволяє ознайомитися з базовим функціоналом системи. Ця роль передбачає доступ до перегляду вакансій та пошуку за фільтрами для ознайомлення з пропозиціями. Також неавторизовані користувачі можуть зареєструватися в системі, щоб отримати розширений доступ до її можливостей.

Шукач роботи після авторизації отримує доступ до функцій, спрямованих на пошук роботи та взаємодію з роботодавцями. Він може створювати й редагувати резюме, надсилати відгуки на вакансії, а також переглядати статус своїх відгуків, що дозволяє контролювати процес працевлаштування.

Роботодавець має функціонал, орієнтований на управління вакансіями та взаємодію з потенційними кандидатами. Ця роль дозволяє створювати й редагувати вакансії, переглядати відгуки, які надходять на ці вакансії, а також отримувати доступ до бази резюме, що допомагає у виборі кандидатів.

Адміністратор володіє розширеними правами, які забезпечують підтримку системи в актуальному та безпечному стані. Адміністратор має можливість керувати користувачами, включаючи зміну їхніх ролей або обмеження доступу, а також займатися модерацією контенту для забезпечення відповідності правил та стандартів системи.

Такий розподіл ролей забезпечує чітку організацію функціоналу відповідно до потреб кожної категорії користувачів.

2.2 Моделювання поведінки системи

Моделювання поведінки системи дозволяє візуалізувати основні процеси її роботи, що допомагає краще зрозуміти функціональність і взаємодію між компонентами. Для цієї мети використовуються UML-діаграми, які забезпечують формалізований підхід до опису динаміки системи.

2.2.1 Діаграма прецедентів

Діаграма прецедентів, як один з основних інструментів моделювання, представляє ключові варіанти використання системи різними типами користувачів. На цій діаграмі зображені ролі (актори), що взаємодіють із системою, а також прецеденти, які відображають їхні дії в рамках доступного функціоналу (див. рис. 2.1).

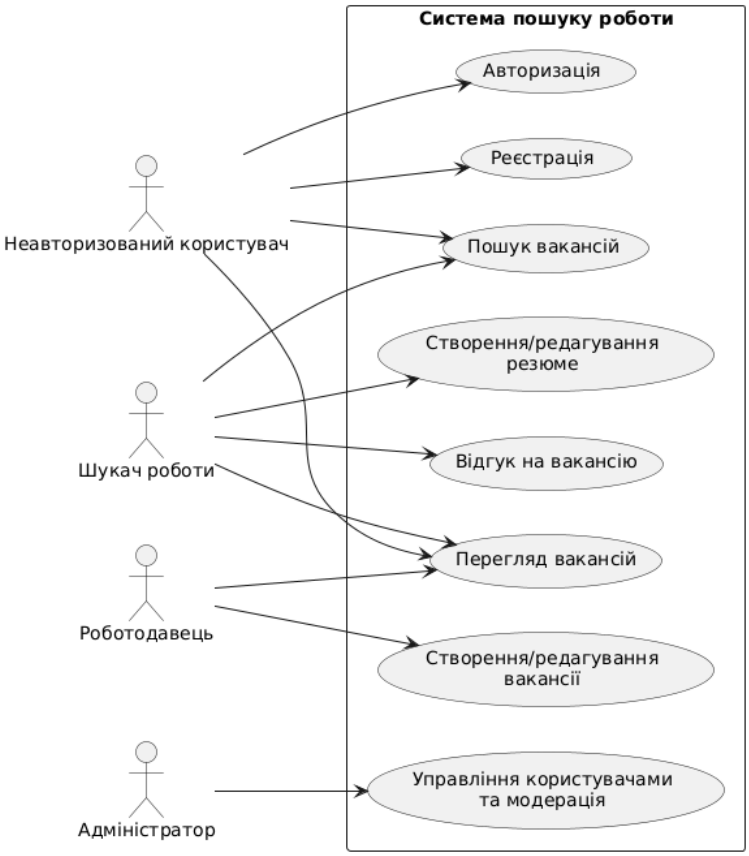


Рисунок 2.1 – Діаграма прецедентів вебзастосунку

Кожен тип користувача представлений окремим актором. Наприклад, неавторизований користувач може переглядати вакансії або зареєструватися в системі. Шукач роботи має доступ до створення резюме та відгуків на вакансії, роботодавець – до управління вакансіями, а адміністратор – до модерації контенту та управління користувачами. Лінії асоціацій між акторами та прецедентами вказують на взаємодії, що виникають у процесі використання системи.

Дана діаграма дозволяє наочно побачити розподіл функціоналу між ролями та визначити, як різні типи користувачів взаємодіють із системою, що є основою для подальшого проєктування архітектури та реалізації компонентів.

2.2.2 Діаграма послідовності створення вакансії

Діаграма послідовності відображає процес створення нової вакансії роботодавцем, деталізуючи взаємодію між користувачем та компонентами системи (див. рис. 2.2).

Процес починається з того, що роботодавець відкриває форму створення вакансії через інтерфейс клієнтської частини. Під час заповнення форми виконується клієнтська валідація введених даних. Цей етап дозволяє перевірити коректність значень у реальному часі, що допомагає уникнути передачі некоректної інформації на сервер.

Після завершення заповнення даних форма відправляє запит до серверної частини. На сервері виконується додаткова валідація отриманих даних, що гарантує їхню відповідність вимогам системи. Якщо всі перевірки проходять успішно, інформація про вакансію зберігається у базі даних, яка використовується для зберігання та подальшого доступу до вакансій.

Завершальним етапом є надсилання підтвердження роботодавцю. Це підтвердження може включати повідомлення про успішне створення вакансії або, у разі виникнення помилок, інструкції щодо їх усунення.

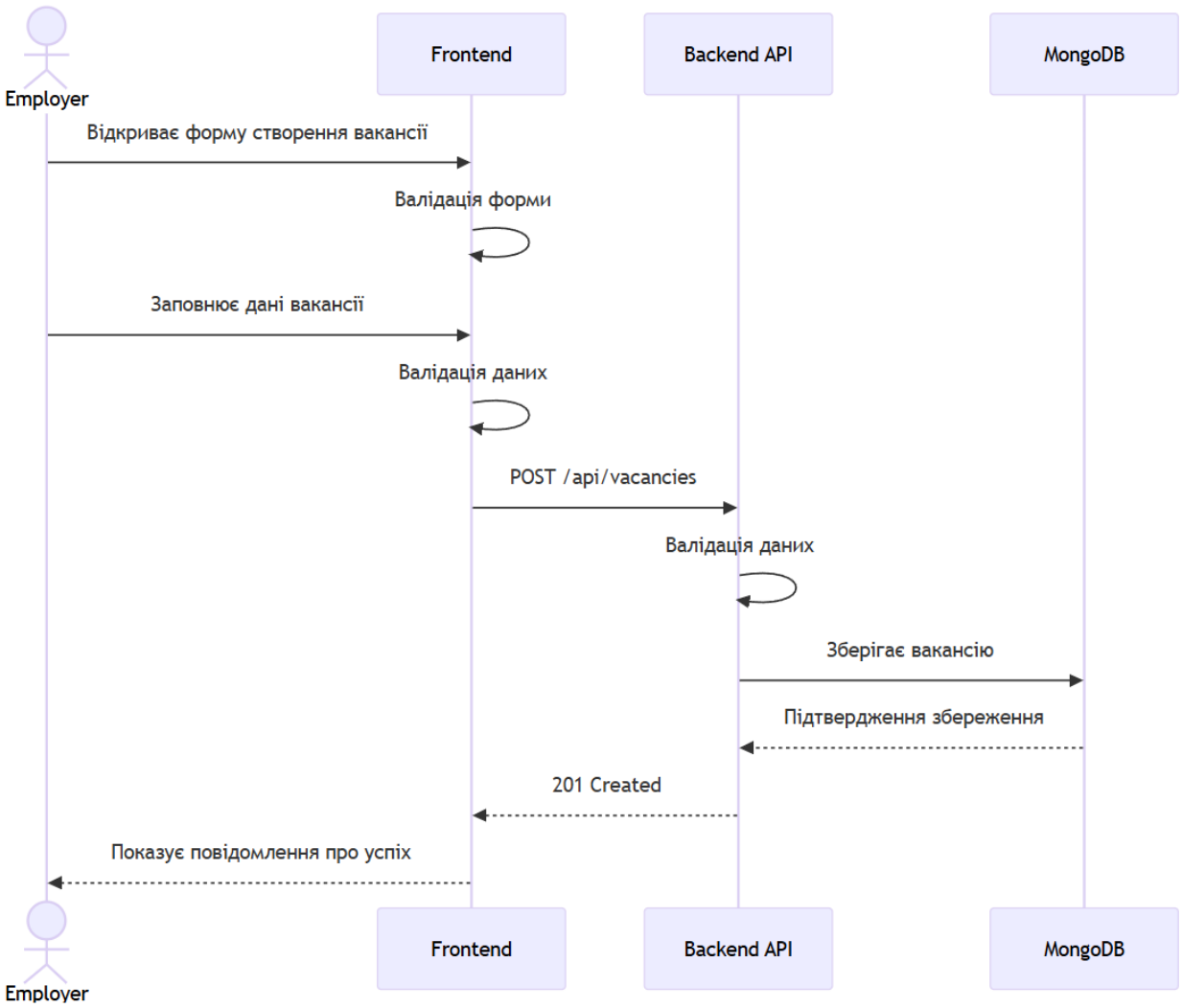


Рисунок 2.2 – Діаграма послідовності створення вакансії

Така послідовність дій забезпечує надійність процесу, перевіряючи дані на кількох рівнях та надаючи користувачеві зворотний зв’язок про результат операції.

2.3 Прототипування та візуалізація користувацького інтерфейсу

Прототипування та візуалізація користувацького інтерфейсу є ключовими етапами розробки вебзастосунку, що спрямовані на створення зрозумілого та зручного інтерфейсу для кінцевих користувачів. Вони дають змогу на ранніх етапах оцінити зручність використання системи, а також виявити потенційні недоліки в дизайні.

На зображенні (див. рис. 2.3) представлений базовий макет сторінки з вакансіями.

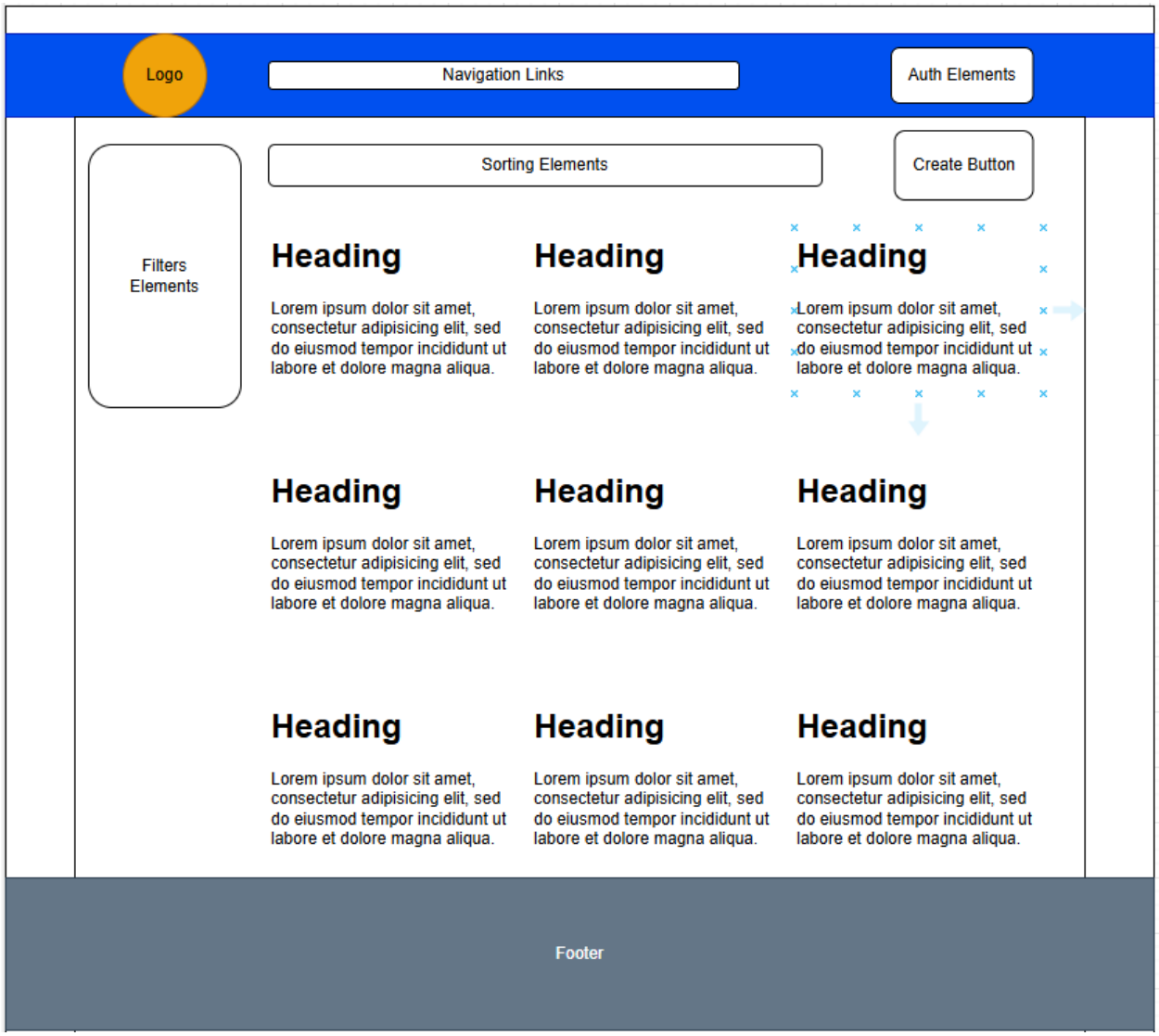


Рисунок 2.3 – Ескіз користувацького інтерфейсу

Цей макет демонструє розташування основних елементів, таких як заголовки, фільтри, список вакансій та кнопки дій. Прототип такого типу дозволяє швидко побачити, як буде виглядати і працювати сторінка, та слугує відправною точкою для подальшого вдосконалення.

Цей ескіз допомагає візуалізувати, як користувачі зможуть взаємодіяти з системою, наприклад, переглядати вакансії, використовувати фільтри для звуження пошуку або переходити до деталей конкретної пропозиції. Він також

полегшує комунікацію між командою розробників та дизайнерами, дозволяючи спільно працювати над удосконаленням інтерфейсу.

Таким чином, прототипування є важливим кроком у створенні інтуїтивно зрозумілого і функціонального користувацького інтерфейсу, що відповідає потребам цільової аудиторії.

2.3.1 Визначення центральних сутностей системи

Визначення центральних сутностей системи дозволяє структурувати дані, які є основою для функціонування вебзастосунку. У цьому проєкті головними сутностями є вакансії та резюме, оскільки вони визначають ключовий зміст і взаємодію користувачів із системою.

Вакансії слугують центральним елементом для роботодавців. Вони містять детальну інформацію про робочі позиції, включаючи опис обов’язків, вимоги до кандидатів, рівень оплати праці, умови зайнятості та місце роботи. Ці дані формують основу для пошуку роботи та є головним об’єктом, який привертає увагу шукачів.

Резюме представляють іншу ключову сутність, орієнтовану на шукачів роботи. Вони містять відомості про професійні навички, досвід роботи, освіту та інші характеристики, які можуть зацікавити роботодавців. Структура резюме розроблена таким чином, щоб забезпечити максимально зручний пошук і вибір кандидатів.

Ці дві сутності є ядром, навколо якого будується весь користувацький інтерфейс і логіка взаємодії системи. Вони визначають основні сценарії використання системи, такі як створення та перегляд вакансій і резюме, пошук відповідних пропозицій чи кандидатів, а також взаємодія між роботодавцями та шукачами роботи.

2.3.2 Створення макетів основних сторінок

Важливий етап проєктування, який визначає ключові елементи інтерфейсу та структуру взаємодії користувачів із системою. У рамках цього процесу розроблено кілька базових сторінок, що охоплюють основні сценарії використання системи.

Головна сторінка є стартовою точкою для користувачів і включає такі елементи, як навігаційне меню для переходу між розділами, логотип системи, а також інструменти аутентифікації. Крім того, на цій сторінці представлені фільтри та інструменти сортування вакансій, що дозволяють налаштовувати пошук за відповідними критеріями. Центральним елементом головної сторінки є список вакансій, який надає користувачам доступ до актуальних пропозицій.

Сторінка вакансії містить детальну інформацію про обрану вакансію. На ній представлений опис позиції, вимоги до кандидатів, умови роботи та контактні дані роботодавця. Це дозволяє користувачам отримувати повне уявлення про пропозицію та, за необхідності, зв’язатися з роботодавцем.

Сторінка резюме забезпечує доступ до детальної інформації про конкретного шукача роботи. Вона включає опис навичок, професійного досвіду, рівня освіти та контактні дані. Ця сторінка слугує основою для роботодавців, які здійснюють пошук кандидатів на відкриті вакансії.

Сторінка створення вакансії надає роботодавцям форму для додавання нових вакансій. Інтерфейс передбачає введення всіх необхідних даних, таких як назва позиції, вимоги, умови праці та контактна інформація. Форма включає клієнтську валідацію для перевірки коректності введених даних.

Сторінка створення резюме орієнтована на шукачів роботи, які бажають додати своє резюме до системи. Вона містить форму для введення інформації про навички, досвід роботи, освіту та інші важливі аспекти, які можуть зацікавити роботодавців.

Розробка цих макетів забезпечує базу для подальшого дизайну інтерфейсу та його реалізації, а також сприяє кращому розумінню структури і функціональності системи.

2.3.3 Візуалізація елементів інтерфейсу

Візуалізація елементів інтерфейсу є важливим етапом у розробці вебзастосунку, оскільки вона визначає зовнішній вигляд і організацію основних компонентів, які забезпечують взаємодію користувачів із системою.

Навігаційне меню розташоване у верхній частині сторінки й слугує основним інструментом для переміщення між розділами застосунку. Воно містить посилання на ключові сторінки, такі як «Головна», «Вакансії», «Резюме» та «Про нас», забезпечуючи користувачам швидкий доступ до основного функціоналу.

Логотип знаходиться в лівому верхньому куті і виконує роль ідентифікатора бренду. Він дозволяє користувачам легко асоціювати застосунок із компанією або проєктом, підвищуючи впізнаваність.

Елементи аутентифікації, зокрема кнопки для входу та реєстрації, розташовані в правому верхньому куті. Вони надають користувачам можливість увійти до системи або створити обліковий запис.

Фільтри та сортування розташовані в лівій частині сторінки, поруч зі списком вакансій. Ці елементи дозволяють користувачам налаштовувати відображення вакансій за обраними параметрами, такими як місце розташування, рівень заробітної плати або тип зайнятості, полегшуючи пошук відповідних пропозицій.

Список вакансій займає центральну частину сторінки й містить карточки з короткою інформацією про кожну вакансію. На карточках відображаються такі дані, як назва посади, місце розташування, рівень заробітної плати та інші ключові параметри, що допомагають швидко оцінити пропозицію.

Кнопка створення, розташована у правому верхньому куті над списком вакансій, дає роботодавцям можливість додавати нові вакансії. Вона чітко виділена, щоб забезпечити зручний доступ до функції створення вакансій.

Footer розташований у нижній частині сторінки та містить додаткову інформацію, зокрема посилання на інші ресурси, політику конфіденційності чи контактні дані. Цей елемент допомагає структурувати інформацію, яка не є центральною для основної взаємодії, але залишається важливою.

Ітеративний підхід до розробки інтерфейсу передбачає регулярний перегляд і вдосконалення елементів на основі зворотного зв’язку від користувачів. Це дозволяє адаптувати інтерфейс до їхніх потреб, покращуючи загальний досвід роботи з системою.

3 РОЗРОБКА, ТЕСТУВАННЯ ТА РОЗГОРТАННЯ ДОДАТКА

3.1 Розробка клієнтської частини (Frontend)

Розробка клієнтської частини (Frontend) базується на використанні React у поєднанні з TypeScript, що дозволяє створити надійний, ефективний і легко масштабований інтерфейс користувача.

Вибір TypeScript як супутньої технології для клієнтської частини зумовлений його численними перевагами. Статична типізація допомагає запобігати помилкам ще на етапі написання коду, значно скорочуючи час на налагодження. Крім того, розширена підтримка IDE з автодоповненням робить процес розробки більш зручним і продуктивним. Система типів TypeScript також сприяє кращій документації коду, полегшуючи читання та розуміння навіть складних проєктів. Завдяки цьому рефакторинг стає простішим, що забезпечує легке внесення змін та підтримку коду в довгостроковій перспективі.

React обрано як основу для розробки через його компонентний підхід, який дозволяє розділяти інтерфейс на окремі багаторазові модулі. Це спрощує як розробку, так і подальшу підтримку проєкту. Ефективність забезпечується завдяки використанню Virtual DOM, який оптимізує оновлення сторінок, зменшуючи навантаження на браузер і підвищуючи швидкість роботи застосунку. Велика екосистема готових рішень і бібліотек, доступних для React, дозволяє швидко інтегрувати додатковий функціонал, скорочуючи час розробки. Крім того, активна спільнота розробників забезпечує широкий доступ до документації, прикладів та підтримки.

Об’єднання можливостей TypeScript і React створює ідеальне середовище для побудови клієнтської частини сучасного вебзастосунку, що забезпечує зручність для користувачів і ефективність для розробників.

3.1.1 Архітектурний підхід

Проєкт ґрунтується на використанні методології Feature-Driven Development (FDD), яка передбачає групування коду за функціональними модулями. Кожен модуль відповідає за окрему частину функціональності системи й об’єднує всі пов’язані компоненти – від стилів і логіки до API-запитів. Такий підхід дозволяє чітко розмежувати відповідальність різних частин застосунку, що забезпечує більш упорядковану структуру (див. рис. 3.1).

```
src/
├── features/          # Функціональні модулі
│   ├── Vacancies/     # Модуль вакансій
│   ├── Resumes/       # Модуль резюме
│   ├── Auth/          # Модуль аутентифікації
│   └── Admin/         # Адміністративний модуль
```

Рисунок 3.1 – Модульна структура проєкту

Однією з ключових переваг FDD є зручна організація коду. Завдяки поділу на функціональні модулі структура проєкту стає логічнішою, що полегшує роботу над великими або довготривалими проєктами. Крім того, цей підхід значно спрощує масштабування, оскільки нові модулі можна додавати без значного впливу на існуючі. Ще однією важливою перевагою є те, що всі файли, пов’язані з певною функцією, знаходяться в одному місці. Це скорочує час на пошук і внесення змін, особливо коли проєкт розробляється командою.

FDD також покращує процес тестування. Окремі модулі можуть перевірятися незалежно від інших частин системи, що знижує ймовірність виникнення помилок та полегшує підтримку проєкту в майбутньому. Такий підхід робить роботу з кодом більш передбачуваною та гнучкою, дозволяючи адаптувати систему до нових вимог або розширювати її функціональність без значних труднощів.

3.1.2 Управління станом

Для управління станом у сучасних вебзастосунках зазвичай використовується Redux – бібліотека, яка забезпечує централізоване зберігання та керування станом додатку [1, 18]. Redux дозволяє організувати дані у передбачуваний спосіб, що особливо важливо для застосунків із великою кількістю взаємодій і компонентів. Але у цьому проєкті я обрав Redux Toolkit – офіційний набір інструментів, який спрощує використання Redux і вирішує низку типових завдань. Redux Toolkit суттєво спрощує централізоване управління станом завдяки інтуїтивному API, що дозволяє зменшити обсяг шаблонного коду. Використання createSlice забезпечує зручне визначення стану та редюсерів. Наприклад, у vacanciesSlice визначено логіку для оновлення списку вакансій та застосування фільтрів. Як це реалізовано наведено на рисунку 3.2.

```
const vacanciesSlice = createSlice({
  name: 'vacancies',
  initialState,
  reducers: {
    setVacancies: (state, action) => {
      state.items = action.payload;
    },
    setFilters: (state, action) => {
      state.filters = action.payload;
    },
    // ... інші редюсери
  },
});
```

Рисунок 3.2 – Приклад управління станом вакансій

Цей фрагмент коду показує, як setVacancies і setFilters дозволяють оновлювати список вакансій та застосовувати фільтри. Такі редюсери

автоматично генерують відповідні дії, спрощуючи управління станом і забезпечуючи чітку структуру коду. Крім того, функція createAsyncThunk дозволяє легко працювати з асинхронними операціями, такими як запити до API, що є особливо корисним для застосунків, які активно взаємодіють із сервером.

Цей підхід робить розробку більш ефективною, забезпечуючи чистоту та узгодженість коду, що є критично важливим для створення масштабованих і підтримуваних систем.

3.1.3 Компонентна структура

Компонентна структура застосунку побудована на основі функціональних компонентів із використанням hooks, що замінили класові компоненти. Цей підхід, детально описаний в [1], надає сучасний та ефективний спосіб організації логіки компонентів, а також забезпечує кращу продуктивність і простоту коду (див. рис. 3.3).

```
export function VacancyDetails() {
  const { id } = useParams<{ id: string }>();
  const [vacancy, setVacancy] = useState<Vacancy | null>(null);

  useEffect(() => {
    const fetchVacancy = async () => {
      const response = await getVacancyById(id);
      setVacancy(response.data);
    };
    fetchVacancy();
  }, [id]);

  // ... рендеринг компонента
}
```

Рисунок 3.3 – Приклад використання hooks у компоненті вакансії

Наприклад, у компоненті VacancyDetails використовується hook useParams для отримання параметрів маршруту та useState для зберігання стану вакансії. Hook useEffect забезпечує виконання асинхронного запиту для отримання даних про вакансію за її ідентифікатором. Код виглядає так, як показано на рисунку 3.3.

Функціональні компоненти мають кілька важливих переваг. По-перше, їхній код є чистішим і легшим для розуміння, оскільки не потрібно використовувати методи життєвого циклу, як у класових компонентах. По-друге, вони забезпечують кращу продуктивність за рахунок меншої ваги і простішої реалізації. Нарешті, hooks дозволяють повторно використовувати логіку, що спрощує організацію коду в складних застосунках.

У результаті використання функціональних компонентів із hooks не лише покращує зручність розробки, але й підвищує ефективність роботи застосунку.

3.1.4 Взаємодія з API

Взаємодія з API у застосунку організована через окремі модулі, що забезпечує чітке розділення логіки роботи з сервером і компонентами інтерфейсу. Це рішення дозволяє створити масштабовану та зручну для підтримки архітектуру, а також спрощує інтеграцію нових функцій.

Наприклад, для отримання вакансій із застосуванням фільтрів використовується модуль api/vacancies.ts. У ньому визначено функцію getVacancies, яка виконує запит до сервера за допомогою бібліотеки Axios [7] (див. рис. 3.4).

Такий підхід має кілька суттєвих переваг. По-перше, обробка помилок реалізована централізовано, що підвищує надійність системи й полегшує підтримку. По-друге, завдяки використанню TypeScript типізація відповідей сервера дозволяє мінімізувати ризики, пов’язані з некоректними даними, і

покращує автодоповнення коду в IDE. По-третє, виділення API-запитів в окремий модуль робить їх легкими для тестування й модифікації без необхідності змінювати основну логіку компонентів.

```
// api/vacancies.ts
export const getVacancies = async (
  params: VacancyFilters
): Promise<ApiResponse<Vacancy[]>> => {
  try {
    const response = await axios.get(`${API_BASE_URL}/vacancies`, { params });
    return { data: response.data };
  } catch (error) {
    return handleApiError(error);
  }
};
```

Рисунок 3.4 – Приклад запиту фільтрованої інформації через API на сервер

Така організація забезпечує зручність і гнучкість у взаємодії з сервером, що є ключовим фактором для розробки й підтримки складних вебзастосунків.

3.2 Розробка серверної частини (Backend)

Розробка серверної частини (Backend) базується на використанні Node.js разом із Express.js як основного фреймворку. Ця комбінація була обрана через її ефективність, гнучкість та широкий спектр можливостей, які вона надає для створення вебсервісів [2].

Node.js забезпечує середовище виконання JavaScript на сервері, що дозволяє використовувати одну мову програмування як на клієнтській, так і на серверній частині застосунку. Це спрощує розробку та зменшує кількість необхідних технологій у проєкті. Крім того, Node.js відзначається високою продуктивністю для обробки I/O операцій, що робить його ідеальним для

вебзастосунків, які активно взаємодіють із базами даних або API. Широка екосистема npm забезпечує доступ до численних готових рішень і бібліотек, а активна спільнота сприяє швидкому вирішенню проблем та вдосконаленню коду.

Express.js було обрано за його гнучкість і мінімалізм. Цей фреймворк дозволяє швидко створювати серверні додатки з простими й зручними засобами для маршрутизації. Завдяки системі middleware можна легко додавати функціонал, як-от аутентифікацію чи обробку запитів. Крім того, Express.js забезпечує легку інтеграцію з MongoDB, що є важливим для організації роботи з базою даних у цьому проєкті.

3.2.1 Архітектура API сервера

Архітектура API серверної частини побудована відповідно до принципів RESTful, що забезпечує стандартизовану взаємодію між клієнтською та серверною частинами через HTTP-методи. Це дозволяє реалізувати основні функціональні можливості застосунку та забезпечує простоту інтеграції й масштабованість (див. рис. 3.5).

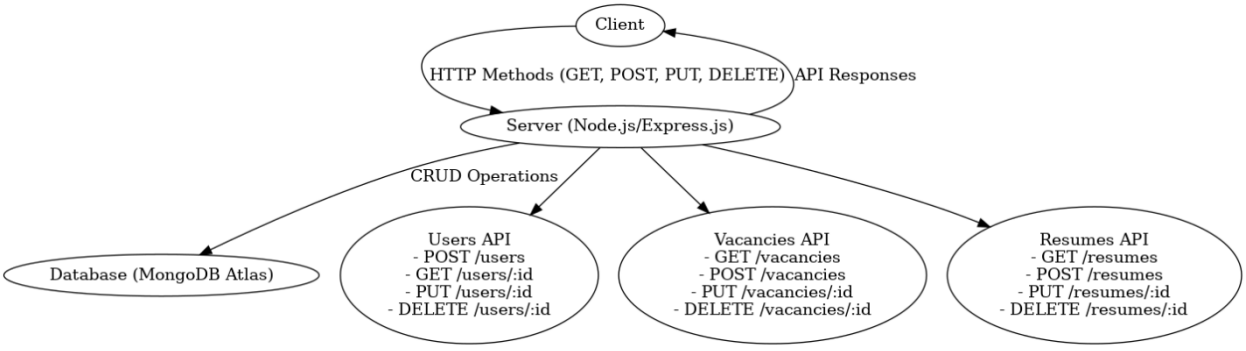


Рисунок 3.5 – Структурна діаграма REST API

Серед ключових функціональних блоків серверної частини можна виділити управління вакансіями та резюме. API забезпечує створення нових записів, їх оновлення та видалення. Наприклад, роботодавці можуть додавати

вакансії або редагувати вже існуючі, тоді як шукачі роботи отримують аналогічні можливості для своїх резюме. Усі ці операції виконуються через відповідні HTTP-методи: POST для створення, PUT для оновлення та DELETE для видалення.

Управління користувачами реалізовано як окремий функціональний блок, що включає реєстрацію, авторизацію та призначення ролей. Користувачі можуть зареєструватися як роботодавці, шукачі роботи чи адміністратори, залежно від своїх потреб. Ця частина API відповідає за збереження безпеки системи та надання доступу до відповідного функціоналу залежно від ролі користувача.

Додатково API забезпечує обробку запитів, пов’язаних із пошуком вакансій або резюме. Наприклад, користувачі можуть застосовувати фільтри для пошуку за місцем розташування, рівнем заробітної плати або іншими параметрами, а також сортувати результати за датою публікації чи рівнем оплати. Ці можливості реалізовані через GET-запити із передачею відповідних параметрів.

3.2.2 Структура серверної частини

Ця частина проєкту організована таким чином, щоб забезпечити зрозуміле розташування файлів і модулів, які виконують окремі функції. Це дозволяє легко розширювати та підтримувати серверну частину, а також сприяє зручності співпраці між розробниками (див. рис. 3.6).

У директорії models знаходяться схеми MongoDB, які описують структуру даних, що зберігаються в базі. Наприклад, тут визначено схеми для вакансій, профілів користувачів і громад, включаючи типи даних та правила валідації.

Файли маршрутизації зберігаються в директорії routes. Вони відповідають за визначення кінцевих точок API, таких як GET /vacancies або POST /users, і направляють запити до відповідних контролерів.

```
/server
  /models        # Схеми MongoDB для вакансій, користувачів, громад
  /routes        # Маршрути для вакансій, користувачів
  /controllers   # Логіка обробки запитів
  /utils         # Утиліти (обробка помилок, middleware)
  server.js      # Основний файл, який ініціалізує сервер
```

Рисунок 3.6 – Файлова структура серверної частини

Усі бізнес-логіка запитів реалізована в директорії controllers. Тут знаходяться функції, які взаємодіють із моделями, обробляють дані й формують відповіді для клієнта.

Директорія utils містить допоміжні модулі, включаючи функції для централізованої обробки помилок, middleware для перевірки автентичності та інші утиліти, які використовуються у різних частинах системи.

Файл server.js є точкою входу для серверної частини. У ньому налаштовується підключення до бази даних, ініціалізується сервер Express і конфігуруються глобальні middleware.

Така структура підтримує чіткість і масштабованість коду, полегшує тестування і розробку нових функціональних можливостей.

3.2.3 Моделі даних

Моделі даних забезпечують структуру для організації й зберігання інформації в базі даних. Однією з ключових моделей у системі є модель для зберігання вакансій (Vacancy), яка відображає основні атрибути, необхідні для функціонування вебзастосунку (див. рис. 3.7).

Серед основних полів моделі виділяються:

– title, що представляє ідентифікатор посади та посилається на колекцію “positions” – це забезпечує узгодженість і можливість використання загальної бази для назв посад;

- location, який є ідентифікатором громади з колекції “localities” – завдяки цьому зберігається зв’язок із географічним розташуванням вакансії;
- salary, що визначає розмір заробітної плати – це поле використовується як одна з ключових характеристик при фільтрації вакансій;
- educationDegree, де вказується ступінь освіти, необхідний для виконання роботи, наприклад, бакалавр чи магістр;
- experience, який зберігає інформацію про досвід роботи в роках;
- postedDate, що автоматично визначає дату публікації вакансії для подальшого сортування та відображення актуальності.

```
const VacancySchema = new mongoose.Schema({
  title: { type: mongoose.Schema.Types.ObjectId, ref: 'Position' },
  location: { type: mongoose.Schema.Types.ObjectId, ref: 'Locality' },
  salary: Number,
  educationDegree: String,
  experience: String,
  postedDate: { type: Date, default: Date.now },
});
```

Рисунок 3.7 – Приклад схеми MongoDB для вакансій

Ця схема використовує посилання (ref) на інші колекції, що дозволяє зберігати дані у взаємозв’язаній структурі. Такий підхід сприяє узгодженості інформації, спрощує пошук і забезпечує ефективність запитів до бази даних. Більш детальний опис моделювання даних наведено в підрозділі 3.3.

3.2.4 Реалізація логіки фільтрації даних

Реалізація логіки фільтрації даних у серверній частині є одним із ключових компонентів для забезпечення персоналізованого пошуку вакансій.

Для цього використовується конвеєр MongoDB Aggregation, який дозволяє динамічно обробляти запити з фільтрами та виконувати складні операції над даними безпосередньо в базі (див. рис. 3.8).

```
router.get('/api/vacancies', async (req, res) => {
  const filter = {};
  if (req.query.title) {
    filter['title'] = new mongoose.Types.ObjectId(req.query.title);
  }
  if (req.query.territorialCommunity) {
    filter['locationDetails.territorialCommunity'] = req.query.territorialCommunity;
  }

  const vacancies = await Vacancy.aggregate([
    { $match: filter },
    {
      $lookup: {
        from: 'positions',
        localField: 'title',
        foreignField: '_id',
        as: 'titleDetails',
      },
    },
    { $unwind: { path: '$titleDetails', preserveNullAndEmptyArrays: true } },
  ]);

  res.status(200).json(vacancies);
});
```

Рисунок 3.8 – Приклад коду маршруту для фільтрації вакансій.

Основні критерії фільтрації включають вибір вакансій за посадою (title), територіальною громадою (territorialCommunity), рівнем освіти (educationDegree) та відсутністю досвіду роботи. Ця логіка реалізована в маршруті GET /api/vacancies, який обробляє запити з відповідними параметрами.

Наведений код демонструє, як налаштовуються динамічні фільтри й використовується конвеєр aggregate для обробки запитів, виконуючи кроки:

- створюється об’єкт filter, який динамічно наповнюється умовами залежно від параметрів запиту;
- за допомогою стадії $match виконується початкове фільтрування вакансій;
- стадія $lookup об’єднує колекцію positions для отримання додаткових даних про посаду;
- $unwind забезпечує розгортання вкладених масивів для зручнішої обробки результатів.

Цей механізм забезпечує гнучкість фільтрації та дозволяє ефективно працювати з великими обсягами даних, повертаючи лише ті результати, які відповідають заданим умовам. Такий підхід покращує продуктивність і спрощує логіку обробки на стороні клієнта.

3.2.5 Тестування API

Тестування API проводилося з використанням Postman [14] – популярного інструменту для мануального тестування RESTful API. Цей інструмент дозволяє створювати, надсилати та аналізувати HTTP-запити, що робить його зручним вибором для перевірки функціональності серверної частини.

Postman використовувався для перевірки ключових маршрутів, таких як створення, оновлення, видалення та отримання вакансій. Наприклад, тестування маршруту GET /api/vacancies включало відправлення запитів із різними параметрами фільтрації для перевірки правильності роботи конвеєра MongoDB Aggregation. Для маршруту POST /api/vacancies перевірялася коректність валідації даних на сервері, а також можливість створення нових записів у базі даних.

Крім базового тестування, у Postman були створені колекції запитів, що включали сценарії для перевірки комплексних взаємодій, таких як авторизація користувачів і обробка помилкових запитів. Це дозволило швидко перевіряти всі основні функції після внесення змін до коду серверної частини.

Завдяки використанню Postman вдалося переконатися, що серверна частина обробляє запити відповідно до вимог і повертає очікувані відповіді. Цей підхід забезпечив початковий рівень тестування перед автоматизованими перевірками, що дозволило швидко виявляти та усувати потенційні помилки в логіці API.

3.2.6 Інтеграція фронтенду та бекенду

Інтеграція фронтенду та бекенду у проєкті здійснювалася за допомогою бібліотеки Axios, яка є зручним інструментом для виконання HTTP-запитів. Axios використовується для взаємодії з RESTful API, створеним на серверній частині, і забезпечує просту реалізацію функціоналу отримання, створення, оновлення чи видалення даних через API.

Приклад API-функції для отримання фільтрованих вакансій показує, як можна легко організувати запити з параметрами, використовуючи бібліотеку Axios (див. рис. 3.9).

Ця функція демонструє кілька ключових можливостей Axios:

- легке передавання параметрів у запиті через об’єкт params, що дозволяє динамічно додавати фільтри для отримання вакансій;
- автоматична обробка JSON-даних, що спрощує обробку відповідей із сервера;
- зручний синтаксис роботи з промісами, що робить код читабельним і зрозумілим.

Серед переваг Axios також слід виділити можливість налаштування заголовків і токенів для авторизації, що дозволяє інтегрувати механізми безпеки, наприклад, Bearer Token для захищених API.

```
import axios from "axios";

export const getVacancies = async (
  page: number,
  limit: number = 18,
  filters?: Record<string, string | number | undefined>
): Promise<ApiResponse<Vacancy[]>> => {
  try {
    const response = await axios.get(`${API_BASE_URL}/api/vacancies`, {
      params: {
        page,
        limit,
        ...filters,
      },
    });
    return { data: response.data };
  } catch (error) {
    console.error("Error fetching vacancies:", error);
    return { data: [], error: error.message };
  }
};
```

Рисунок 3.9 – Приклад API-функції для отримання фільтрованих вакансій

Цей інструмент забезпечує ефективну інтеграцію між клієнтською та серверною частинами, дозволяючи швидко адаптувати функціонал до змін у бізнес-логіці. Крім того, Axios спрощує процес обробки помилок, що сприяє стабільній роботі застосунку та зменшує час, необхідний для тестування та виправлення помилок.

3.3 Проєктування бази даних

Проєктування бази даних для вебзастосунку з пошуку роботи базується на використанні MongoDB Atlas, що було обрано як основне хмарне рішення. Цей вибір обґрунтований низкою переваг, які сприяють як зручності розробки, так і ефективності підтримки системи.

MongoDB Atlas надає переваги хмарного розміщення, що дозволяє уникнути складнощів із налаштуванням і підтримкою власних серверів. Автоматичне резервне копіювання забезпечує безпеку даних, а можливість масштабування гарантує стабільну роботу бази навіть під час пікових навантажень. Інтегровані інструменти моніторингу спрощують контроль стану бази, що є особливо важливим для великих проєктів.

Гнучкість схеми MongoDB дозволяє зберігати документи з різною структурою, що є корисним для даних, які часто змінюються. Додавання нових полів можливе без необхідності виконання складних міграцій, що робить базу адаптивною до змін у бізнес-логіці. Такий підхід особливо зручний у випадках, коли структура вакансій чи резюме може змінюватися залежно від потреб користувачів.

Формат JSON, у якому зберігаються дані, дозволяє легко працювати з MongoDB у середовищі JavaScript/TypeScript, завдяки чому розробка проходить швидше та ефективніше. Багатий набір операторів для запитів сприяє створенню складних запитів без значних зусиль, що робить базу універсальним рішенням для зберігання вакансій, резюме та пов’язаної інформації.

Загалом, MongoDB Atlas забезпечує необхідний баланс між гнучкістю, зручністю роботи та продуктивністю, що є ключовим для динамічних застосунків із великим обсягом даних.

3.3.1 Структура колекцій

Для зберігання даних у проєкті були спроєктовані основні колекції: “positions” (посади), “localities” (населені пункти), “employers” (роботодавці) та “vacancies” (вакансії). Ці колекції організовані так, щоб забезпечити стандартизацію даних і підтримку фільтрації, а також встановлення взаємозв’язків між різними сутностями (див. рис. 3.10).

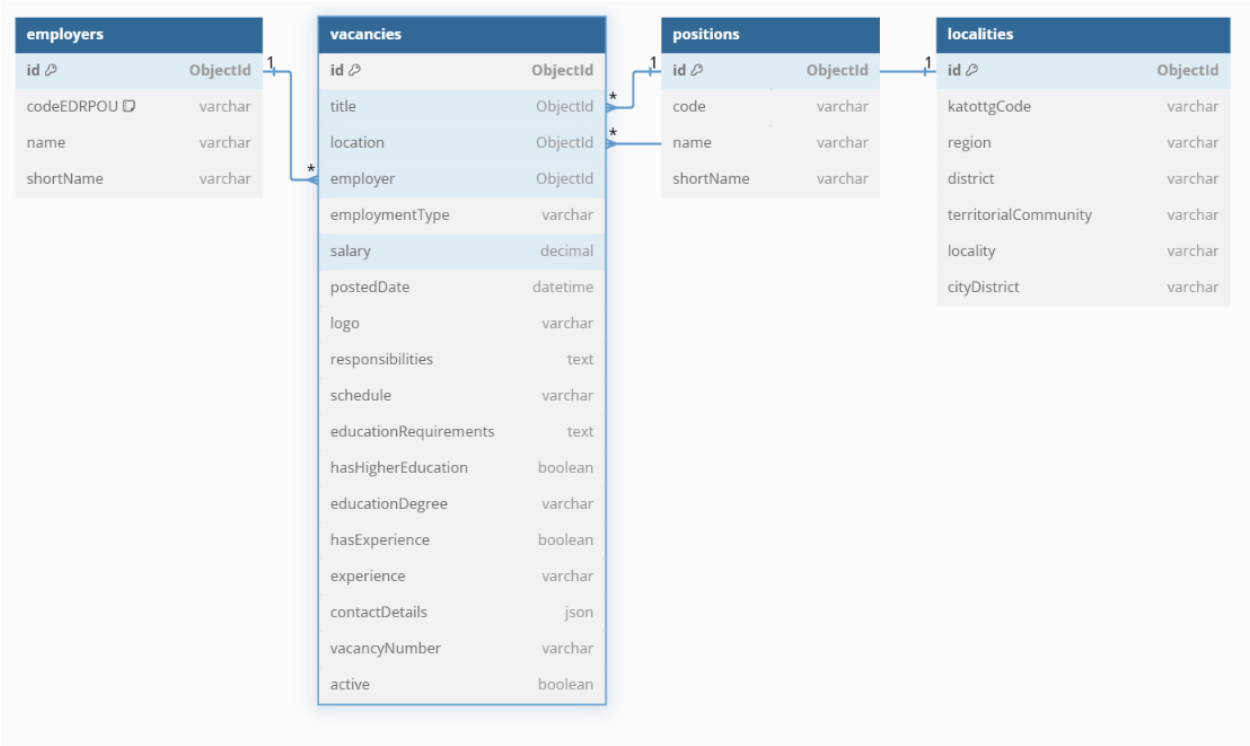


Рисунок 3.10 – Колекція вакансій і її зв’язки

Колекція “positions” зберігає довідник посад, що дозволяє стандартизувати назви вакансій і полегшувати пошук. Колекція “localities” виконує роль географічного довідника, містячи інформацію про населені пункти, громади та регіони. У колекції “employers” зберігається інформація про роботодавців, включаючи їхні назви, коди ЄДРПОУ та статус активності.

Основною колекцією є “vacancies”, яка зберігає інформацію про пропозиції роботи. Вона має посилання на інші колекції: “positions” для визначення посади, “localities” для зазначення місця роботи та “employers” для ідентифікації роботодавця. Ця організація забезпечує узгодженість даних і підтримку складних запитів із фільтрацією за регіонами, типом зайнятості чи рівнем заробітної плати.

3.3.2 Початкове наповнення бази даних

Початкове наповнення бази даних виконувалося за допомогою спеціально розроблених скриптів імпорту, які обробляють дані з наявних

файлів у форматі .csv і масово додають їх до відповідних колекцій у базі даних. Це дозволяє швидко завантажувати великі обсяги інформації, забезпечуючи автоматизацію процесу та уникнення помилок, характерних для ручного введення даних (див. рис. 3.11).

Для імпорту даних про вакансії був створений скрипт, який використовує бібліотеку для роботи з CSV-файлами. Дані обробляються рядок за рядком, проходять валідацію та зберігаються в базі. Наприклад, поле «Посада» шукається у колекції “positions”, а за відсутності відповідного запису створюється новий. Аналогічно обробляються дані про роботодавців і населені пункти.

Приклад реалізації скрипту наведено на рисунку 3.11.

```
const importVacanciesFromCSV = async () => {
  try {
    fs.createReadStream('../data/vacancies.csv')
      .pipe(csvParser({ separator: ';' }))
      .on('data', async (row) => {
        const vacancyData = {
          title: await findOrCreatePosition(row['Посада']),
          employer: await findOrCreateEmployer(row['Роботодавець']),
          location: await findLocality(row['Населений пункт']),
          // ... інші поля
        };
        await Vacancy.create(vacancyData);
      });
  } catch (error) {
    console.error('Помилка імпорту:', error);
  }
};
```

Рисунок 3.11 – Приклад скрипту імпорту вакансій

Скрипт виконує такі основні дії:

– читає файл vacancies.csv через потік;
– обробляє кожен рядок, перетворюючи його в об’єкт із відповідними полями;

- використовує допоміжні функції (findOrCreatePosition, findOrCreateEmployer, findLocality) для пошуку або створення пов’язаних записів у колекціях;
- зберігає підготовлені дані у базу через модель Vacancy.

Цей підхід дозволяє зберігати узгодженість даних між колекціями завдяки автоматичному створенню відсутніх записів і валідації інформації. Завдяки автоматизації вдалося суттєво скоротити час первинного наповнення бази та забезпечити її якість.

3.3.3 Оптимізація та індексація

Оптимізація та індексація є важливим аспектом для забезпечення ефективності роботи бази даних, особливо в системах із великими обсягами даних і складними запитами. У проєкті були створені індекси для колекції vacancies, що дозволяють значно прискорити пошук і фільтрацію (див. рис. 3.12).

```
// Індексація ключових полів
db.vacancies.createIndex({ "title": 1 });
db.vacancies.createIndex({ "location": 1 });
db.vacancies.createIndex({ "employer": 1 });
db.vacancies.createIndex({ "salary": 1 });
db.vacancies.createIndex({ "postedDate": -1 });
db.vacancies.createIndex({
  "active": 1,
  "hasHigherEducation": 1,
  "hasExperience": 1
});

// Текстовий індекс для пошуку
db.vacancies.createIndex({
  "responsibilities": "text",
  "educationRequirements": "text"
});
```

Рисунок 3.12 – Індексування колекції вакансій

Для забезпечення швидкого доступу до даних були створені індекси за ключовими полями. Наприклад, поле title (посада) індексується для швидкого пошуку вакансій за позиціями, а location (місцезнаходження) дозволяє ефективно знаходити вакансії за географічним розташуванням. Індекси за полями employer і salary забезпечують швидкий доступ до даних роботодавців і можливість фільтрації за рівнем оплати.

Для підтримки сортування вакансій за датою публікації було створено індекс за полем postedDate, який зберігає записи у зворотному порядку. Додатковий комбінований індекс охоплює такі параметри, як статус вакансії (active), наявність вищої освіти (hasHigherEducation) і досвіду роботи (hasExperience). Це оптимізує багатокритеріальну фільтрацію вакансій.

Також для текстового пошуку був створений текстовий індекс, що охоплює поля responsibilities (обов’язки) та educationRequirements (вимоги до освіти). Це дозволяє виконувати пошук за ключовими словами, забезпечуючи більш гнучкі можливості для користувачів.

Індекси значно скорочують час виконання запитів до бази даних, забезпечуючи:

- швидкий пошук і фільтрацію даних за ключовими полями;
- ефективну підтримку складних умов і сортування;
- можливість текстового пошуку за кількома полями.

Завдяки цій оптимізації система забезпечує швидку реакцію на запити користувачів, навіть у випадках великого навантаження або роботи з великими наборами даних.

3.4 Структура вебзастосунку

3.4.1 Клієнтська частина та її файлова структура

Файлова структура клієнтської частини побудована так, щоб забезпечити логічне розташування коду й полегшити розробку. Усі

функціональні модулі згруповані у папку features, де кожен модуль має свою ізольовану структуру. Наприклад, модуль Vacancies містить усі компоненти, пов’язані з відображенням вакансій, включаючи фільтри, сортування, створення нових вакансій та деталі кожної вакансії. Таке групування дозволяє легко знайти необхідний компонент або логіку (див. рис. 3.13).

```
client/
├── public/                # Статичні файли
├── src/                   # Основний код додатку
│   ├── api/               # Взаємодія з backend API
│   │   ├── config.ts      # Конфігурація API (endpoints, base URL)
│   │   ├── types.ts       # TypeScript типи для API responses
│   │   └── vacancies.ts   # API методи для роботи з вакансіями
│   │
│   ├── features/          # Функціональні модулі додатку
│   │   ├── About/         # Модуль "Про нас"
│   │   │   └── About.tsx
│   │   ├── Resumes/       # Модуль резюме
│   │   │   └── Resumes.tsx
│   │   └── Vacancies/     # Модуль вакансій
│   │       ├── CreateVacancy.tsx    # Створення вакансії
│   │       ├── VacancyCard.tsx      # Картка вакансії
│   │       ├── VacancyDetails.tsx   # Деталі вакансії
│   │       ├── VacancyFilters.tsx   # Фільтри вакансій
│   │       ├── VacancySorts.tsx     # Сортування вакансій
│   │       ├── Vacancies.tsx        # Головний компонент вакансій
│   │       └── vacanciesSlice.ts    # Redux slice для вакансій
│   │
│   ├── hooks.ts           # Кастомні React hooks
│   ├── store.ts           # Redux store конфігурація
│   ├── App.tsx            # Головний компонент додатку
│   └── index.tsx          # Точка входу в додаток
│
├── package.json           # Залежності та скрипти
└── tsconfig.json          # Конфігурація TypeScript
```

Рисунок 3.13 – Структура фронтенду

Взаємодія з сервером реалізована в папці api, де зберігаються функції для роботи з backend API, конфігурація та типізація відповідей. Це забезпечує чіткий розподіл відповідальності між функціональними частинами коду.

Глобальний стан додатку зберігається у Redux Store, налаштування якого розташоване у файлі store.ts. Для кожного модуля створюється окремий slice, як-от vacanciesSlice, що забезпечує централізоване управління станом.

Маршрутизація організована за допомогою React Router і відображає чітку вкладену структуру маршрутів (див. рис. 3.14). Головний компонент App обгорнутий у Provider, що забезпечує доступ до глобального стану через Redux у всьому додатку. Основні маршрути, такі як /about, /vacancies, та /resumes, мають вкладені маршрути, наприклад, для перегляду деталей вакансій чи створення нової вакансії.

```
const router = createBrowserRouter([
  {path: '/',
    element: <Provider store={store}><App /></Provider>,
    children: [
      {path: '/about', element: <About />,},
      {path: '/vacancies', element: <Vacancies />,},
      {path: '/resumes', element: <Resumes />,},
      {path: '/vacancies/:id', element: <VacancyDetails />},
      {path: '/vacancies/create', element: <CreateVacancy />},
    ],
  },
]);
```

Рисунок 3.14 – Структура марштрутів

Централізована маршрутизація спрощує управління навігацією та полегшує масштабування застосунку при додаванні нових функцій.

Реалізація побудована на функціональних компонентах із використанням React hooks, що забезпечує чіткий розподіл відповідальності. Кожен компонент має вузьку спеціалізацію, наприклад, VacancyCard

відповідає лише за відображення картки вакансії, а VacancyFilters реалізує логіку фільтрації. Такий підхід робить компоненти перевикористовуваними, зменшуючи дублікат коду.

Використання TypeScript додає рівень безпеки, забезпечуючи типізацію пропсів, стану та API-запитів, що знижує кількість помилок і полегшує підтримку коду.

3.4.2 Логіка серверної частини

Логіка серверної частини побудована на основі багатошарової архітектури з чітким поділом відповідальності між її складовими (див. рис. 3.15). Такий підхід дозволяє легко масштабувати систему, забезпечувати її модульність та підтримувати чистоту коду.

```
server/
├── controllers/     # Обробники запитів
├── models/          # Mongoose моделі
├── routes/          # API маршрути
├── middleware/      # Middleware функції
├── imports/         # Імпорт даних
└── exports/         # Експорт даних
```

Рисунок 3.15 – Структура бекенду

Структура серверної частини включає кілька ключових елементів.

Контролери відповідають за реалізацію бізнес-логіки та обробку запитів клієнтів. Логіка контролерів розділена так, щоб кожен метод виконував лише одну чітко визначену задачу. Наприклад, у контролері вакансій реалізовано пошук вакансії за ідентифікатором із додатковою агрегацією даних (див. рис. 3.16). При цьому контролери обробляють помилки, гарантуючи, що клієнт отримує коректні відповіді на запити.

```
export const getVacancyById = async (req, res) => {
  try {
    const vacancy = await Vacancy.aggregate([
      { $match: { _id: new mongoose.Types.ObjectId(req.params.id) } },
      ...buildLookupStages(),
    ]);
    if (!vacancy.length) {
      return res.status(404).json({ message: 'Vacancy not found' });
    }
    res.json(vacancy[0]);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

Рисунок 3.16 – Приклад контролера вакансії

Моделі даних (директорія models) визначають структуру колекцій у базі MongoDB. Для кожної сутності створено схеми, які описують атрибути, типи даних і правила валідації. Зв’язки між колекціями реалізовані через референції, що забезпечує узгодженість даних та оптимальність запитів.

Маршрути (директорія routes) є точкою входу для HTTP-запитів і перенаправляють їх до відповідних контролерів. Це дозволяє підтримувати чітку структуру API й забезпечує легкість у підтримці та розширенні функціональності.

Middleware-функції обробляють запити на рівні маршрутизації, додаючи додаткові шари безпеки та функціональності. Наприклад, middleware для аутентифікації перевіряє, чи авторизований користувач має доступ до потрібного ресурсу.

Імпорт і експорт даних реалізовані через окремі скрипти, які дозволяють швидко оновлювати базу та підтримувати її актуальність.

Серверна частина підтримує широкий функціонал, включаючи аутентифікацію, авторизацію, механізми валідації даних та обробки помилок.

Значну увагу приділено оптимізації продуктивності завдяки індексації бази даних, ефективним запитам через агрегації, а також впровадженню систем моніторингу та логування. Це забезпечує стабільність, безпеку та надійність системи для кінцевих користувачів.

3.5 Етапи розробки вебзастосунку

Розробка вебзастосунку була реалізована через послідовність взаємопов'язаних етапів. На першому етапі було створено базовий каркас застосунку, що включав ініціалізацію клієнтської частини за допомогою Create React App із TypeScript та налаштування серверного середовища на базі Node.js з Express.js і MongoDB Atlas.

Другий етап зосередився на реалізації базової маршрутизації. На клієнтській частині впроваджено React Router для навігації між основними сторінками, тоді як серверна частина отримала базові CRUD-ендпоінти для роботи з даними.

Третій етап охопив розробку ключових функціональних модулів. На фронтенді реалізовано компоненти для відображення списків вакансій, резюме та форм введення. Серверна частина доповнилася відповідними обробниками для всіх основних сутностей системи.

Четвертий етап забезпечив інтеграцію клієнтської та серверної частин через Axios із типізацією відповідей за допомогою TypeScript. Впроваджено централізовану систему аутентифікації та обробки помилок.

П'ятий етап впровадив глобальне управління станом через Redux Toolkit, що забезпечило ефективну синхронізацію даних між компонентами системи.

Шостий етап присвячено тестуванню та оптимізації, включаючи API-тестування через Postman, оптимізацію бази даних та впровадження механізмів кешування.

Завершальний етап розширив функціональність системи, додавши можливості сортування, розширеного пошуку та фільтрації даних.

3.5.1 Використані бібліотеки та утиліти

Спираючись на сучасні практики веброзробки [1, 2], у процесі створення застосунку було використано комплекс технологій та інструментів. Клієнтська частина базується на React з TypeScript, використовуючи Material-UI [11] для створення адаптивного інтерфейсу. Управління станом реалізовано через Redux [18], а взаємодія з сервером – через Axios [7]. Обробка форм здійснюється за допомогою React Hook Form [15], а маршрутизація – через React Router [17].

Серверна частина побудована на Node.js з Express.js, використовуючи Mongoose для взаємодії з MongoDB. Конфігурація середовища керується через dotenv, а безпека крос-доменних запитів забезпечується CORS.

Процес розробки підтримується Git [8] та GitHub [9] для версіонування, ESLint та Prettier для стандартизації коду, а Nodemon забезпечує зручність розробки. Для роботи з даними використовується CSV-Parser, а тестування API проводиться через Postman.

3.5.2 Створення базового каркасу проєкту

Початковий етап розробки зосередився на створенні фундаментальної структури проєкту. Клієнтська частина отримала чітку організацію компонентів та модулів через create-react-app з TypeScript, а базовий інтерфейс реалізовано за допомогою Material-UI.

Серверна частина структурована навколо Express.js з чітко визначеними моделями даних через Mongoose. Розроблено схеми для основних сутностей

системи: вакансій, роботодавців та локацій, з урахуванням специфіки предметної області.

Інтеграційний шар забезпечив безпечну взаємодію через CORS, а початкове наповнення бази даних реалізовано через спеціалізовані скрипти імпорту CSV-даних. Така архітектура створила надійний фундамент для подальшого розвитку системи.

3.5.3 Розширення функціоналу

У процесі подальшого вдосконалення вебзастосунку, після формування базової архітектури, розробка зосередилась на розширенні функціональних можливостей та оптимізації існуючих компонентів. Особливістю даного етапу став паралельний розвиток як клієнтської, так і серверної частин, що забезпечило планомірне нарощування функціоналу відповідно до визначених бізнес-вимог.

На рівні клієнтської частини було впроваджено низку ключових компонентів, спрямованих на покращення користувацької взаємодії. Зокрема, розроблено систему відображення списку вакансій, що включає як загальний огляд у форматі карток з базовою інформацією, так і розширений перегляд деталей кожної позиції. Імплементовано інтуїтивно зрозумілий інтерфейс створення нових вакансій, доповнений комплексною системою фільтрації та сортування за різними параметрами, включаючи географічне розташування, рівень заробітної плати та форму зайнятості.

Значну увагу було приділено оптимізації користувацького досвіду через впровадження механізму нескінченного скролінгу, що забезпечує плавне довантаження контенту при прокручуванні сторінки. Адаптивність інтерфейсу досягнуто завдяки використанню компонентів Material-UI, що гарантує коректне відображення на пристроях з різною роздільною здатністю. Проведено суттєву оптимізацію продуктивності через усунення надлишкових API-запитів та рефакторинг програмного коду.

Серверна частина зазнала розширення API-функціоналу для підтримки нових можливостей клієнтського інтерфейсу. Реалізовано складну систему динамічної фільтрації, що дозволяє комбінувати різні параметри пошуку. Впроваджено оптимізовані агрегаційні запити до MongoDB для ефективного пошуку та обробки даних. Розроблено гнучку систему сортування результатів за часовими та числовими показниками.

Процес інтеграції нового функціоналу супроводжувався комплексним тестуванням API за допомогою інструменту Postman. Взаємодія між клієнтською та серверною частинами забезпечувалась через бібліотеку Axios, що дозволило оперативно верифікувати коректність обміну даними та за необхідності оптимізувати логіку обробки запитів.

Таким чином, реалізовані вдосконалення суттєво підвищили якість користувацького досвіду, зробивши вебзастосунок більш функціональним та зручним у використанні. Впровадження розширеного набору інструментів для роботи з даними та адаптивного інтерфейсу дозволило ефективно задовольнити потреби цільової аудиторії при збереженні високої продуктивності системи.

3.5.4 Тестування та відлагодження

В рамках забезпечення якості програмного забезпечення було реалізовано комплексний підхід до тестування та відлагодження вебзастосунку, що охоплював як мануальну перевірку користувацького інтерфейсу, так і програмне тестування серверного API. Даний етап розробки мав критичне значення для виявлення та усунення потенційних проблем, оптимізації продуктивності та забезпечення надійного функціонування системи.

Мануальне тестування зосереджувалося на верифікації користувацької взаємодії з системою через різні сценарії використання. Особлива увага

приділялася коректності відображення компонентів у різних браузерах, адаптивності дизайну на різних пристроях, функціональності навігації між розділами застосунку. Важливим аспектом було тестування валідації форм та механізмів фільтрації даних, що забезпечують основний функціонал системи.

Для тестування серверного API було розроблено спеціалізований інструментарій, що включав як використання професійних засобів тестування Postman, так і власні тестові функції (див. рис. 3.17).

```
const testVacanciesEndpoint = async () => {
  try {
    const response = await axios.get(`${API_BASE_URL}/api/vacancies`, {
      params: {title: "someTitle", hasHigherEducation: true, sort: "salary_desc"}
    });
    console.log("Test passed:", response.status === 200);
    console.log("Data structure valid:", validateVacancyStructure(response.data));
  } catch (error) {
    console.error("Test failed:", error.message);
  }
};
```

Рисунок 3.17 – Приклад тестування API для отримання вакансій

Показовим прикладом слугує функція тестування ендпоінту отримання вакансій, яка перевіряє коректність обробки параметрів запиту та валідність структури отриманих даних.

В процесі тестування було виявлено та усунено ряд проблем продуктивності. Зокрема, проведено оптимізацію агрегаційних запитів до MongoDB шляхом реструктуризації етапів агрегації для зменшення навантаження на базу даних. Також впроваджено механізм контролю дублювання запитів з використанням React useRef hook, що запобігає надмірним зверненням до сервера в межах одного циклу оновлення компонента.

Для ефективного відлагодження застосовувався комплекс спеціалізованих інструментів: Chrome DevTools для аналізу DOM-структури та мережевої активності, React Developer Tools для моніторингу стану компонентів, Redux DevTools для контролю глобального стану застосунку. Додатково використовувались Postman для тестування REST API, Lighthouse для оцінки продуктивності та доступності інтерфейсу, MongoDB Compass для оптимізації запитів до бази даних.

Реалізований комплексний підхід до тестування та відлагодження дозволив досягти високого рівня стабільності та продуктивності системи. Більше того, створено фундамент для подальшого впровадження автоматизованого тестування та розширеної системи моніторингу функціонування застосунку.

3.5.5 Розгортання застосунку

На завершальному етапі розробки вебзастосунку було реалізовано процес розгортання системи з використанням сучасних хмарних платформ. Для забезпечення оптимальної продуктивності та надійності було прийнято рішення розділити розгортання на дві частини: серверну частину розміщено на платформі Heroku [3], а клієнтську – на Google Firebase [6].

Вибір Heroku як платформи для розгортання бекенду обґрунтовано її потужними можливостями щодо управління Node.js застосунками. Heroku надає комплексне середовище для розгортання та масштабування серверних додатків, що включає автоматизоване управління залежностями та конфігурацією середовища виконання. Особливо цінною виявилася можливість безпосередньої інтеграції з GitHub, що дозволило налаштувати автоматичне розгортання оновлень при кожному внесенні змін до репозиторію проєкту.

Для розміщення клієнтської частини застосунку було обрано Firebase Hosting від Google, що є спеціалізованим сервісом для розгортання

вебзастосунків. Ключовою перевагою даної платформи стала вбудована підтримка CDN (Content Delivery Network), яка забезпечує оптимальну швидкість доставки контенту користувачам незалежно від їх географічного розташування. Firebase Hosting також надає вбудовані механізми безпеки, включаючи автоматичну підтримку SSL/TLS сертифікатів та захист від DDoS-атак.

Важливим аспектом вибору Firebase стала можливість подальшої інтеграції з додатковими сервісами екосистеми Google, такими як Firebase Authentication для управління користувачами та Firestore для роботи з даними. Це створює потенціал для майбутнього розширення функціональності застосунку без необхідності значних змін в інфраструктурі.

Поточна версія вебзастосунку знаходиться за посиланням: https://job-search-app-32ff3.web.app/vacancies, а інтерфейс зображено на скріншоті (див. рис. 3.18).

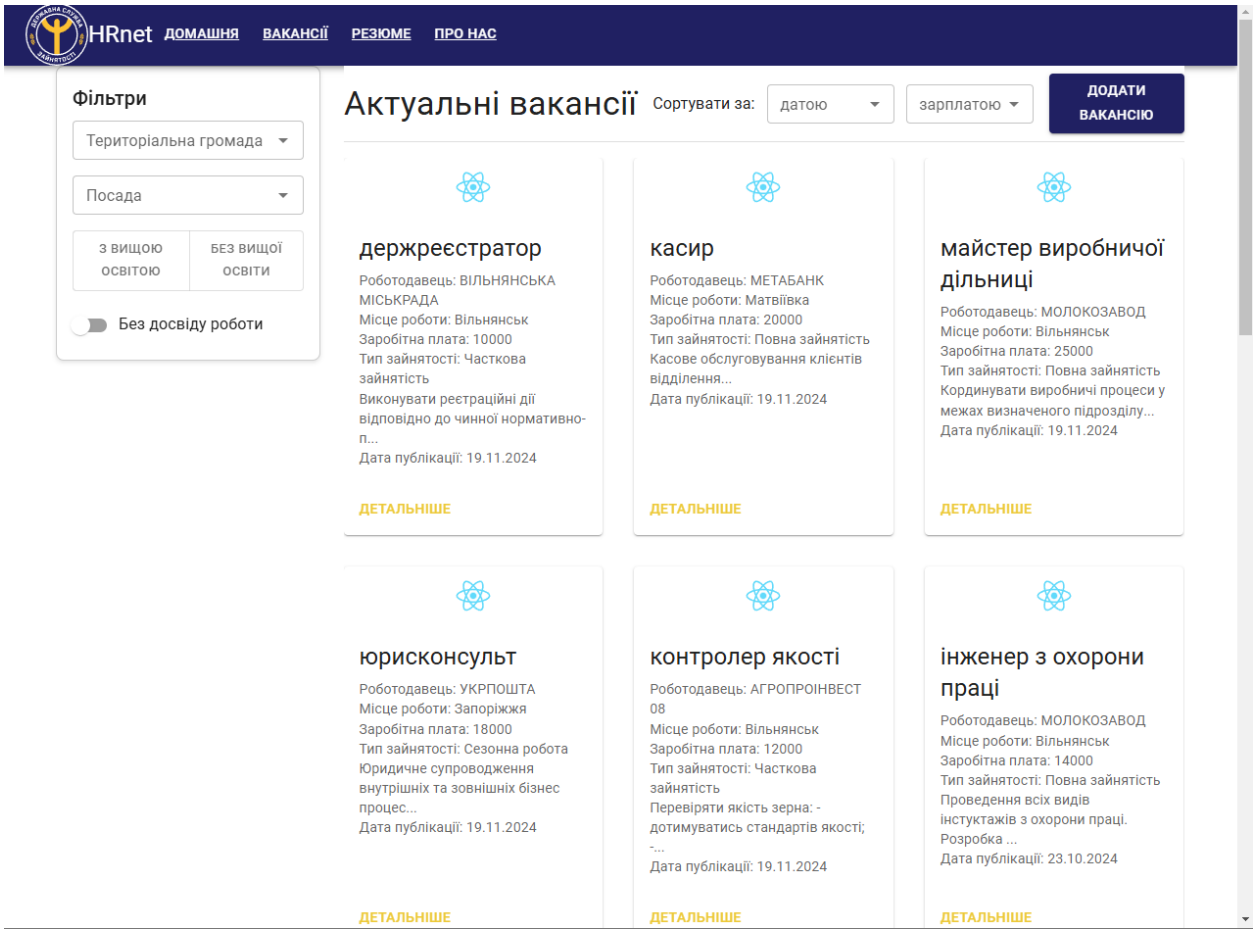


Рисунок 3.18 – Скріншот користувацького інтерфейсу

Обрана архітектура розгортання забезпечує оптимальний баланс між продуктивністю, масштабованістю та безпекою. Використання спеціалізованих платформ для різних компонентів системи дозволяє максимально ефективно використовувати їх переваги, одночасно мінімізуючи затрати на адміністрування та підтримку інфраструктури. Реалізована схема розгортання також створює надійний фундамент для подальшого розвитку та масштабування проєкту відповідно до зростаючих потреб користувачів.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було проведено дослідження та практичну реалізацію вебзастосунку для автоматизації процесів працевлаштування з використанням сучасного технологічного стеку MERN. В процесі роботи здійснено огляд актуального стану ринку праці та обґрунтовано доцільність розробки спеціалізованого програмного рішення для підтримки процесів пошуку роботи та підбору персоналу.

Дослідження можливостей MERN стеку продемонструвало його ефективність для розробки сучасних вебзастосунків. Використання MongoDB як бази даних забезпечило необхідну гнучкість у роботі зі схемами даних та ефективну обробку різних типів інформації. Комбінація Express.js та Node.js дозволила створити надійну серверну платформу, а застосування React надало інструментарій для розробки сучасного користувацького інтерфейсу.

В рамках роботи розроблено архітектуру системи, яка забезпечує ефективну організацію та обробку даних, оптимізований пошук вакансій за різними критеріями та підтримує можливості подальшого розширення функціоналу. Реалізована серверна частина на базі Node.js та Express.js демонструє ефективність у обробці запитів та надійність взаємодії з базою даних MongoDB. Клієнтська частина, створена з використанням React, надає користувачам сучасний адаптивний інтерфейс з інтерактивними компонентами для пошуку та фільтрації вакансій.

Практична реалізація проєкту підтвердила, що сучасні вебтехнології, зокрема MERN стек, надають доступні та ефективні інструменти для створення професійних вебзастосунків, що відповідають індустріальним стандартам. Широка документація та активна спільнота розробників, наявність готових компонентів та бібліотек, а також можливість швидкого прототипування суттєво спрощують процес розробки. Використання єдиної мови програмування (JavaScript/TypeScript) для всіх компонентів системи та

наявність зручних інструментів для розробки й тестування забезпечують ефективність процесу розробки.

Створений вебзастосунок демонструє практичну можливість розробки сучасних рішень для автоматизації процесів працевлаштування з використанням доступних технологій веброзробки. Гнучкість у модифікації структури даних та можливість додавання нових функціональних модулів створюють основу для подальшого розвитку та вдосконалення системи відповідно до зростаючих потреб користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Аналітична та статистична інформація | Державна служба зайнятості. URL: https://www.dcz.gov.ua/stat (дата звернення: 28.06.2024).
2. Banks A., Porcello E. Learning React: Functional Web Development with React and Redux. O'Reilly Media, 2017. 350 p.
3. Brown E. Web Development with Node and Express: Leveraging the JavaScript Stack. O'Reilly Media, Incorporated, 2019. 329 p.
4. Cloud Application Platform | Heroku. URL: https://www.heroku.com (дата звернення: 26.10.2024).
5. Comparing Popular Web Stacks: MERN, MEAN, MEVN, MENG, LAMP, and Ruby on Rails. *Airbrake Blog*. URL: https://blog.airbrake.io/comparing-popular-web-stacks-mern-mean-mevn-meng-lamp-and-ruby-on-rails (дата звернення: 01.07.2024).
6. Express – Node.js web application framework. URL: https://expressjs.com (дата звернення: 30.08.2024).
7. Firebase | Google's Mobile and Web App Development Platform. URL: https://firebase.google.com (дата звернення: 29.10.2024).
8. Getting Started | Axios Docs. URL: https://axios-http.com/docs/intro (дата звернення: 18.10.2024).
9. Git. URL: https://git-scm.com (дата звернення: 10.09.2024).
10. GitHub · Build and ship software on a single, collaborative platform. URL: https://github.com (дата звернення: 10.09.2024).
11. TypeScript: JavaScript With Syntax For Types. URL: https://www.typescriptlang.org (дата звернення: 20.09.2024).
12. Material UI: React components that implement Material Design. URL: https://mui.com/material-ui (дата звернення: 23.09.2024).
13. MongoDB: The Developer Data Platform. URL: https://www.mongodb.com (дата звернення: 23.08.2024).

14. Node.js – Run JavaScript Everywhere. URL: https://nodejs.org (дата звернення: 28.08.2024).
15. Postman: The World’s Leading API Platform. URL: https://www.postman.com (дата звернення: 21.10.2024).
16. React Hook Form – performant, flexible and extensible form library. URL: https://react-hook-form.com (дата звернення: 11.10.2024).
17. React. URL: https://react.dev (дата звернення: 07.08.2024).
18. React Router Official Documentation. URL: https://reactrouter.com (дата звернення: 26.09.2024).
19. Redux – A JS library for predictable and maintainable global state management | Redux. URL: https://redux.js.org (дата звернення: 04.10.2024).
20. The Evolution of Full Stack Development – From LAMP to MERN // GeeksforGeeks. URL: https://www.geeksforgeeks.org/the-evolution-of-full-stack-development-from-lamp-to-mern (дата звернення: 04.07.2024).