МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ВЕБ-ДОДАТКУ ОБЛІКУ КОМП'ЮТЕРІВ І КОМПЛЕКТУЮЧИХ В ОРГАНІЗАЦІЇ»**

Виконав: студент 4 курсу, групи 6.1219-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
А.О. Моторнюк
(ініціали та прізвище)
Керівник декан математичного факультету, професор, д.т.н. Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

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)

“ 07 ” 02 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Моторнюку Артему Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка веб-додатку обліку комп’ютерів і комплектуючих в організації

керівник роботи Гоменюк Сергій Іванович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік питань до розробки.
3. Основна література.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі, аналіз предметної області.
2. Аналіз вимог до інформаційної системи.
3. Реалізація інформаційної системи.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 07.02.2023 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.	08.02.2023	
2.	Збір вихідних даних.	22.02.2023	
3.	Обробка методичних та теоретичних	15.03.2023	
	джерел.		
4.	Розробка першого та другого розділу.	12.04.2023	
5.	Розробка третього розділу та четвертого	15.05.2023	
	розділу.		
6.	Оформлення та нормоконтроль	01.06.2023	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	22.06.2023	

Студент _____ А.О. Моторнюк
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.І. Гоменюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка веб-додатку обліку комп’ютерів і комплектуючих в організації»: 54 с., 12 рис., 10 джерел.

БРАУЗЕР, БАЗА ДАНИХ, ІНТЕРНЕТ, САЙТ, JAVASCRIPT, MVC, NEST JS, POSTGRESQL, REACT, TYPESCRIPT.

Об’єкт дослідження – розробка сайту з використанням React та NestJS.

Мета роботи: розробити веб-додаток для обліку комп’ютерів і комплектуючих в організації.

Методи дослідження – проаналізувати предметні області, вивчення та узагальнення, моделювання.

У кваліфікаційній роботі досліджено предметну область, засоби реалізації веб-додатку обліку комп’ютерів і комплектуючих в організації, обрано найкращий засіб реалізації та розроблено веб-додаток. Розглянуто основні особливості JavaScript, React та NestJS реалізовано базу даних у PostgreSQL.

Клієнтська та серверна частини були розділені з використанням різних технологій, клієнтська частина реалізована з використанням React, серверна частина реалізована з використанням NestJS та працює по принципу запит – відповідь, реалізований зручний інтерфейс.

SUMMARY

Bachelor’s qualifying paper «Development of a web-application for accounting computers and components in the organization»: 54 pages, 12 figures, 10 references.

BROWSER, DATA BASE, INTERNET, WEBSITE, JAVASCRIPT, MVC, NEST JS, POSTGRESQL, REACT, TYPESCRIPT.

The object of the study is website development using React and NestJS.

The aim of the study is to develop a web application for accounting for computers and components in the organization.

The methods of research are to analyze subject areas, study and generalization, modeling.

The qualification work investigates the subject area, means of implementing a web application for accounting for computers and components in an organization, selects the best implementation tool, and develops a web application. The main features of JavaScript, React, and NestJS were considered, and the database was implemented in PostgreSQL.

The client and server parts were separated using different technologies, the client part was implemented using React, the server part was implemented using NestJS and works on the principle of request-response, and a user-friendly interface was implemented.

ЗМІСТ

ВСТУП

В сучасному світі інформаційні технології є невід'ємною частиною будь-якої організації, що дозволяє їм ефективно працювати та досягати поставлених цілей. Однією з найважливіших задач, яку ставлять перед собою багато компаній, є ведення обліку комп'ютерів та комплектуючих. Для цього необхідно мати зручний та функціональний інструмент, який дозволяє відстежувати кількість, стан та розташування всіх комп'ютерів та комплектуючих, що є у власності компанії.

Сутність наукової проблеми полягає в необхідності забезпечення ефективного та надійного обліку інформації про наявність та стан комп'ютерів та комплектуючих в організації.

З огляду на швидкий темп зростання технологій, важливість обліку комп'ютерного обладнання та його комплектуючих зростає щодня. Це особливо актуально для великих організацій зі значною кількістю комп'ютерів, які потребують постійного обслуговування та поновлення.

З іншого боку, багато компаній все ще використовують застарілі методи обліку, такі як ведення таблиць у електронних документах або на паперових носіях. Ці методи не тільки неефективні, але й не надійні, оскільки можуть призвести до втрати важливої інформації та втрати часу на її відновлення.

Таким чином, головною проблемою є розробка ефективного та надійного веб-додатка для обліку комп'ютерів та комплектуючих в організації, який забезпечує швидкий доступ до інформації та покращить процес управління комп'ютерним обладнанням в цілому.

Актуальність дослідження полягає в тому, що в сучасному бізнес-середовищі інформаційні технології є необхідним елементом для ефективної роботи будь-якої організації. Управління комп'ютерною технікою та її облік займають важливе місце в діяльності більшості підприємств. З метою забезпечення ефективної роботи підприємства необхідно мати зручні та

функціональні засоби для обліку комп'ютерної техніки та комплектуючих.

Існуючі програмні рішення для обліку комп'ютерів та комплектуючих часто мають обмежені можливості та не задовольняють потреби бізнесу. Одним з найбільш поширених методів обліку є ведення Excel-таблиць, що не є ефективним та зручним. Відсутність зручного та функціонального програмного забезпечення може призвести до порушення роботи організації через втрату контролю над комп'ютерною технікою та комплектуючими.

Отже, розробка веб-додатка обліку комп'ютерів та комплектуючих є актуальною проблемою, яка вирішить низку проблем управління комп'ютерною технікою та забезпечить ефективну роботу організації.

Для розробки веб-додатка обліку комп'ютерів і комплектуючих в організації було проведено огляд наукових джерел з даної теми. Виявлено, що проблема обліку комп'ютерів та комплектуючих в організаціях є досить актуальною та вимагає вирішення, оскільки розвиток технологій у цій галузі відбувається дуже швидко, а кількість комп'ютерів та комплектуючих у компаніях зростає.

Були виявлені певні розробки в галузі обліку комп'ютерів та комплектуючих, які забезпечують зручність та швидкість процесу, проте деякі з них вимагають великої кількості знань з програмування та налаштування серверів. Тому, розробка простого та зручного веб-додатка, що містить всі необхідні функції для обліку комп'ютерів та комплектуючих у організації, буде актуальною та корисною для багатьох компаній.

Метою дослідження є розробка веб-додатка для обліку комп'ютерів та комплектуючих в організації з метою покращення ефективності управління комп'ютерними ресурсами та забезпечення максимальної доступності даних про інвентар компанії.

Для досягнення цієї мети передбачені наступні завдання:

а) розробити функціональні вимоги до веб-додатка;

б) розробити дизайн та інтерфейс користувача веб-додатка;

в) розробити базу даних для зберігання інформації про комп'ютери та

комплектуючі;

г) реалізувати функціонал для додавання, редагування та видалення даних про комп'ютери та комплектуючі;

д) реалізувати функціонал для пошуку та фільтрації даних;

е) забезпечити захист даних та авторизацію користувачів веб-додатка;

ж) провести тестування веб-додатка та оцінити його ефективність та придатність для використання в організації.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Предметна область

Предметна область «Розробка веб-додатка обліку комп'ютерів і комплектуючих в організації» є досить широкою і затребуваною в сучасному бізнесі.

Основна мета такого веб-додатку – забезпечення ефективного контролю та обліку комп'ютерів та комплектуючих у підприємствах. Це може бути корисним для різних типів організацій, таких як IT-компанії, школи, університети, бібліотеки та інші.

Основні функції веб-додатка можуть включати в себе створення бази даних з детальною інформацією про комп'ютери та комплектуючі, їх серійні номери, дати покупки, дати технічного обслуговування та ремонту, вартість і так далі. Також можливо додавати функції моніторингу технічного стану обладнання та автоматичного повідомлення про несправності.

Для розробки такого веб-додатка необхідні знання та досвід у сфері веб-розробки, баз даних та програмування. Також може бути корисними знання у сфері адміністрування комп'ютерних мереж та системних адміністраційних інструментів.

Аналізуючи предметну область, можна зробити висновок про її високу актуальність у сучасному бізнесі та великий потенціал для розвитку та впровадження.

Основні компоненти предметної області можуть бути описані наступним чином:

- веб-розробка: це процес створення веб-додатків, що включає в себе розробку функціональності та дизайну веб-сторінок, налаштування веб-сервера та бази даних;
- облік комп'ютерів: це процес ведення обліку комп'ютерів та іншого

обладнання в організації, що може включати в себе відслідковування серійних номерів, дати придбання, місця розташування, стану, технічних характеристик тощо;

- облік комплектуючих: це процес ведення обліку комплектуючих до комп'ютерів в організації, таких як процесори, материнські плати, жорсткі диски, пам'ять, відеокарти тощо;
- організація: це контекст, в якому відбувається розробка веб-додатка, і включає в себе визначення вимог та потреб користувачів, налаштування безпеки та доступу до даних, організацію робочих процесів, забезпечення роботи веб-сервера та інфраструктури.

Оскільки веб-додаток призначений для ведення обліку комп'ютерів та комплектуючих в організації, можна зробити висновок, що він буде містити функціонал для реєстрації нових комп'ютерів та комплектуючих, оновлення інформації про існуюче обладнання, відслідковування змін та переміщень обладнання, забезпечення.

1.2 Технічне завдання

Розробити клієнт-серверну архітектуру, призначену для функціонування веб-додатку призначений для ведення обліку комп'ютерів та комплектуючих в організації. Вивчити предметну область, врахувати переваги та недоліки існуючих ринкових рішень.

Функціональні вимоги до клієнтської частини сайту:

а) забезпечити можливість реєстрації користувачів в системі з різними рівнями доступу;
б) реалізувати можливість додавання та видалення інформації про комп'ютери та комплектуючі в базі даних;
в) забезпечити можливість пошуку комп'ютерів та комплектуючих за різними параметрами (наприклад, за назвою, серійному номеру, датою

придбання тощо);

г) реалізувати можливість перегляду детальної інформації про кожен комп'ютер або комплектуючу, включаючи інформацію про характеристики, стан, історію обслуговування та ремонту;

д) забезпечити можливість виділення комп'ютерів або комплектуючих в окремі групи, наприклад, за місцем знаходження, працівником, який з ними працює, тощо;

е) реалізувати можливість створення звітів про наявність комп'ютерів та комплектуючих в організації за різними критеріями (наприклад, за станом, групою, датою останнього обслуговування тощо);

ж) забезпечити можливість редагування та оновлення інформації про комп'ютери та комплектуючі в базі даних;

з) забезпечити можливість створення резервних копій бази даних та її відновлення у разі потреби.

1.3 Вибір технологій

Будь-який багатофункціональний додаток, що працює за принципом клієнт-серверної архітектури, складається з двох основних компонентів: клієнтської частини (frontend) і серверної частини (backend). Кожна з цих частин відповідає за виконання своїх завдань та взаємодію між ними забезпечує роботу додатку в цілому.

Клієнтська частина (frontend):

– інтерфейс користувача: розробляється з використанням технологій, таких як HTML, CSS і JavaScript, для створення зручного інтерфейсу, з якого користувачі зможуть взаємодіяти з додатком;

– клієнтська логіка: включає обробку подій користувача, валідацію даних, відправку запитів до сервера та обробку отриманих відповідей.

Серверна частина (backend):

- серверний фреймворк: використовуючи такі технології, як Node.js, Python (за допомогою фреймворків, наприклад, Flask або Django) або Java (за допомогою фреймворків, наприклад, Spring), розробляється серверна логіка для обробки запитів від клієнта;
- база даних: зазвичай використовується Система Керування Базами Даних (СКБД), така як PostgreSQL, для збереження і керування даними, що використовуються додатком;
- бізнес-логіка: реалізується для обробки запитів, взаємодії з базою даних, валідації даних, автентифікації та авторизації користувачів, обробки логічних операцій і багато іншого.

Забезпечення взаємодії між клієнтською і серверною частинами здійснюється за допомогою комунікаційного протоколу, такого як HTTP або WebSocket. Клієнтська частина взаємодіє з сервером, відправляючи HTTP-запити для отримання або відправлення даних. Серверна частина обробляє ці запити, виконує відповідні операції (наприклад, зчитування чи запис даних в базу даних) і повертає відповіді до клієнта.

Окрім того, багатофункціональний додаток може також включати додаткові компоненти, наприклад:

- аутентифікація та авторизація: забезпечує контроль доступу до функціональності додатку, ідентифікацію користувачів та управління їх правами;
- конфігурація і розгортання: додаток може вимагати налаштування параметрів, таких як URL бази даних, налаштування безпеки, ключі API тощо (також потрібно розгорнути додаток на веб-сервері або хмарній платформі);
- тестування: розробка тестових сценаріїв і виконання їх для перевірки функціональності, продуктивності та безпеки додатку;
- масштабованість: забезпечення можливості розширення додатку, враховуючи зростаюче навантаження і обсяг даних.

У сучасних технологіях виділяються три рівні абстракції.

Рівень інфраструктури (*Infrastructure Level*). Цей рівень охоплює апаратну інфраструктуру, таку як сервери, мережеві комунікації, зберігання даних та інші фізичні компоненти. На цьому рівні зазвичай вирішуються питання, пов'язані з розміщенням, масштабуванням та управлінням інфраструктурою.

Рівень платформи (*Platform Level*). Цей рівень надає середовище для розробки і виконання програмного забезпечення. Він включає в себе платформи розробки додатків (наприклад, веб-фреймворки, середовища виконання), сервіси для керування даними, розгортанням, моніторингом та іншими операціями, що спрощують розробку програмного забезпечення.

Рівень додатків (*Application Level*). Цей рівень стосується конкретних додатків і функціональності, яку вони надають. На цьому рівні розробляються програмні додатки, які використовують платформу і інфраструктуру для забезпечення певної функціональності для користувачів. Це можуть бути веб-додатки, мобільні додатки, системи управління базами даних, інтегровані системи тощо.

Ці три рівні абстракції дозволяють розподілити розробку та управління технологічним стеком на окремі шари, що спрощує розробку, масштабування та управління складними системами. Кожен рівень має свої особливості, інструменти та відповідальності, і разом вони створюють повноцінне середовище для розробки та використання сучасних технологій.

Проєкти прийнято розділяти на наступні три групи.

Проєкти з підтримкою бізнесу (*Business Projects*). Це проєкти, спрямовані на досягнення конкретних бізнес-цілей і вирішення проблем, що виникають у підприємстві або організації. Ці проєкти зазвичай пов'язані зі стратегічним розвитком, покращенням ефективності бізнес-процесів, впровадженням нових продуктів або послуг.

Проєкти з розробки програмного забезпечення (*Software Development Projects*). Це проєкти, спрямовані на створення нового програмного забезпечення або модифікацію та вдосконалення існуючих програмних продуктів. Вони включають в себе всі етапи життєвого циклу розробки ПЗ, від аналізу вимог і

проєктування до реалізації, тестування та впровадження.

Інфраструктурні проєкти (*Infrastructure Projects*). Це проєкти, пов'язані з побудовою, розширенням або модернізацією інфраструктури, необхідної для функціонування організації або проєктів з розробки програмного забезпечення. Вони включають в себе такі аспекти, як мережева інфраструктура, серверне обладнання, системи зберігання даних, безпеку та інші технічні складові.

Ці три групи проєктів допомагають визначити специфіку та особливості кожного проєкту, а також встановити відповідні принципи управління та планування для досягнення успіху.

Проаналізувавши технічне завдання, наш додаток можна віднести до групи «Проєкти з розробки програмного забезпечення» (Software Development Projects). Додаток буде включати різноманітні функції, такі як реєстрація комп'ютерів, ведення журналу обслуговування, контроль за наявністю комплектуючих, аналіз даних і звіти. Ви будете розробляти клієнт-серверну архітектуру, де клієнтська частина буде представлена веб-інтерфейсом, доступним для користувачів через веб-браузер, а серверна частина буде відповідальна за обробку запитів, збереження даних та взаємодію з базою даних.

Для максимально швидкої і ефективної розробки додатка, доцільно використовувати існуючі бібліотеки, фреймворки та компоненти. Дамо більш детальну характеристику поняттю «фреймворк».

Фреймворк – це набір підготовлених компонентів, бібліотек, інструментів і правил, які надають структуру та основу для розробки програмного забезпечення. Він надає розробникам заздалегідь визначену архітектуру та набір функціональності, що допомагає спростити процес розробки та прискорює створення програмних продуктів.

Основні характеристики фреймворків включають:

- структура та організація: фреймворк визначає структуру проєкту, включаючи розподіл файлів, правила іменування, шаблони проєктування та логіку розташування компонентів;
- готові компоненти: фреймворк надає набір готових компонентів і

модулів, які можна використовувати для реалізації загальних функцій, таких як обробка форм, автентифікація, маршрутизація тощо (це дозволяє розробникам уникнути повторного винаходження колеса і зосередитися на унікальних аспектах своєї програми);

- інструменти розробки: фреймворк надає набір інструментів і сервісів, які полегшують розробку, такі як відладчик, генератори коду, система керування залежностями, міграції бази даних тощо (вони допомагають автоматизувати рутинні завдання та покращують продуктивність розробника);
- правила та стандарти: фреймворк накладає правила і стандарти на структуру коду, організацію файлів, стиль програмування та інші аспекти розробки (це дозволяє забезпечити єдність та консистентність в проєкті, полегшує співпрацю розробників та покращує зрозумілість коду).

Загалом, фреймворк є основою для розробки програмного забезпечення, яка надає заздалегідь визначену структуру, компоненти та інструменти для спрощення розробки та прискорення процесу створення програмних продуктів. Використання фреймворку дозволяє розробникам зосередитися на унікальних аспектах свого проєкту, забезпечуючи при цьому ефективність, стабільність та повторне використання коду.

1.4 Мова програмування

У сучасному світі програмування існує безліч мов програмування, кожна з яких має свої особливості, переваги та застосування. Вибір мови програмування є важливим етапом розробки, оскільки від нього залежить продуктивність, розширюваність та якість створеного додатку. На рисунку 1.1 зображено популярні мови програмування.

Language	Community Size (in millions)	Popular Use	Least Popular In…
JavaScript	17.4	Apps, Web	DS/ML, Embedded
Python	15.7	DS/ML, IoT	Mobile, Web
Java	14.0	Cloud, Mobile	DS/ML, Web
C/C++	11.0	Embedded, IoT	Web, Cloud
C#	10.0	Desktop, Games	DS/ML, Mobile
PHP	7.9	Web, Cloud	DS/ML, Mobile
Kotlin	5.0	Mobile, AR/VR	DS/ML, Desktop
Visual Development Tools	5.0	AR/VR, Desktop	Cloud, Web
Swift	3.5	Mobile, AR/VR	Cloud, Embedded
Go	3.3	Cloud, Apps	Mobile, DS/ML
Objective-C	2.4	AR/VR, IoT	Desktop, Apps
Rust	2.2	AR/VR, IoT	Web, Mobile
Ruby	2.1	IoT, Apps	Web, Embedded
Dart	1.8	Mobile, Apps	Web, Apps
Lua	1.4	Games, IoT	Mobile, Embedded

Рисунок 1.1 – Популярні мови веб-програмування

Нижче наведений перелік найпопулярніших мов програмування. Кожна з цих мов має свої особливості, переваги та використовується для різних цілей. Розглянемо їх детальніше.

JavaScript є однією з найпоширеніших мов програмування веб-розробки. Вона забезпечує можливість створювати динамічний та інтерактивний контент на веб-сторінках. JavaScript використовується як мова скриптів у браузері, де вона здатна маніпулювати HTML-елементами, обробляти події, взаємодіяти з сервером і багато іншого.

Яскравий приклад – Facebook.

Python – це мова програмування загального призначення, яка відома своєю простотою та зрозумілістю синтаксису. Вона має багатий набір бібліотек і

фреймворків, які сприяють швидкому розвитку веб-додатків. Python також широко використовується у сфері науки, обробки даних та штучного інтелекту.

Яскравим прикладом є: Instagram, Uber, Dropbox.

PHP – це мова програмування, спеціально призначена для розробки веб-додатків. Вона добре підтримує роботу з базами даних та інтеграцію з веб-серверами. PHP є однією з найпоширеніших мов у сфері веб-розробки і використовується в багатьох популярних фреймворках, таких як Laravel та Symfony.

Яскравим прикладом є Stackoverflow.

Ruby – це приємна для програміста мова програмування, яка покликана підвищити продуктивність та зручність розробки. Вона відома своєю простотою та елегантністю, а також багатим екосистемою гемів (бібліотек) і фреймворків, таких як Ruby on Rails.

Яскравими прикладами є Netflix та Twitter.

Ми розглянули деякі з найпопулярніших мов програмування і описали їх особливості та застосування. Кожна з цих мов має свої переваги і дозволяє розробникам втілювати різноманітні проєкти залежно від їх потреб і вимог.

Для написання цього додатку було вирішено використовувати таку мову програмування як JavaScript. Вибір JavaScript як мови програмування для розробки даного додатку є обґрунтованим з кількох причин.

Веб-орієнтованість: JavaScript є мовою, спеціально створеною для веб-розробки. Завдяки своїй вбудованій підтримці веб-браузерів, JavaScript дозволяє створювати динамічні та інтерактивні елементи на веб-сторінках. Це дозволить нам реалізувати зручний та ефективний інтерфейс користувача для нашого додатку.

Широке сприйняття: JavaScript є однією з найпопулярніших мов програмування, і його використання широко поширене серед розробників. Це означає, що знаходити кваліфікованих розробників JavaScript буде значно легше, а також наявність великої кількості ресурсів, бібліотек і фреймворків, які спрощують розробку веб-додатків.

Розширюваність: JavaScript має велику кількість розширень і бібліотек, які дозволяють розширювати його функціональність та забезпечувати більше можливостей. Наприклад, фреймворки, такі як React.JS, дозволяють побудувати потужний та ефективний інтерфейс користувача. Також, існують багато бібліотек для роботи з базами даних, мережевими запитами та іншими аспектами веб-розробки, що полегшує процес розробки нашого додатку.

Переносимість: JavaScript підтримується більшістю сучасних веб-браузерів, що робить його переносимим із однієї платформи на іншу. Це означає, що додаток буде працювати на різних операційних системах та пристроях, що забезпечує його доступність для широкого кола користувачів.

Таким чином, обрання JavaScript як мови програмування для додатку є логічним рішенням, оскільки вона надає нам потужні інструменти для розробки веб-додатку з інтерактивним інтерфейсом та підтримкою різних платформ.

1.5 Фреймворк для клієнтської частини

У цьому розділі ми розглянемо найпопулярніші JavaScript фреймворки, які використовуються для розробки клієнтської частини веб-додатків. Кожен з цих фреймворків має свої унікальні особливості і привабливі риси, які варто враховувати при виборі.

На рисунку 1.2 ми можете побачити графік популярності запитів до трьох найпопулярніших JavaScript фреймворків.

React.js є одним з найпопулярніших фреймворків для розробки користувацького інтерфейсу. Він базується на компонентній архітектурі, що дозволяє побудувати складний інтерфейс з малими пере використанням компонентів [5]. React.js також використовує віртуальний DOM для ефективного оновлення інтерфейсу. Прикладами систем, що використовують React.js, є Facebook, Instagram, Airbnb.

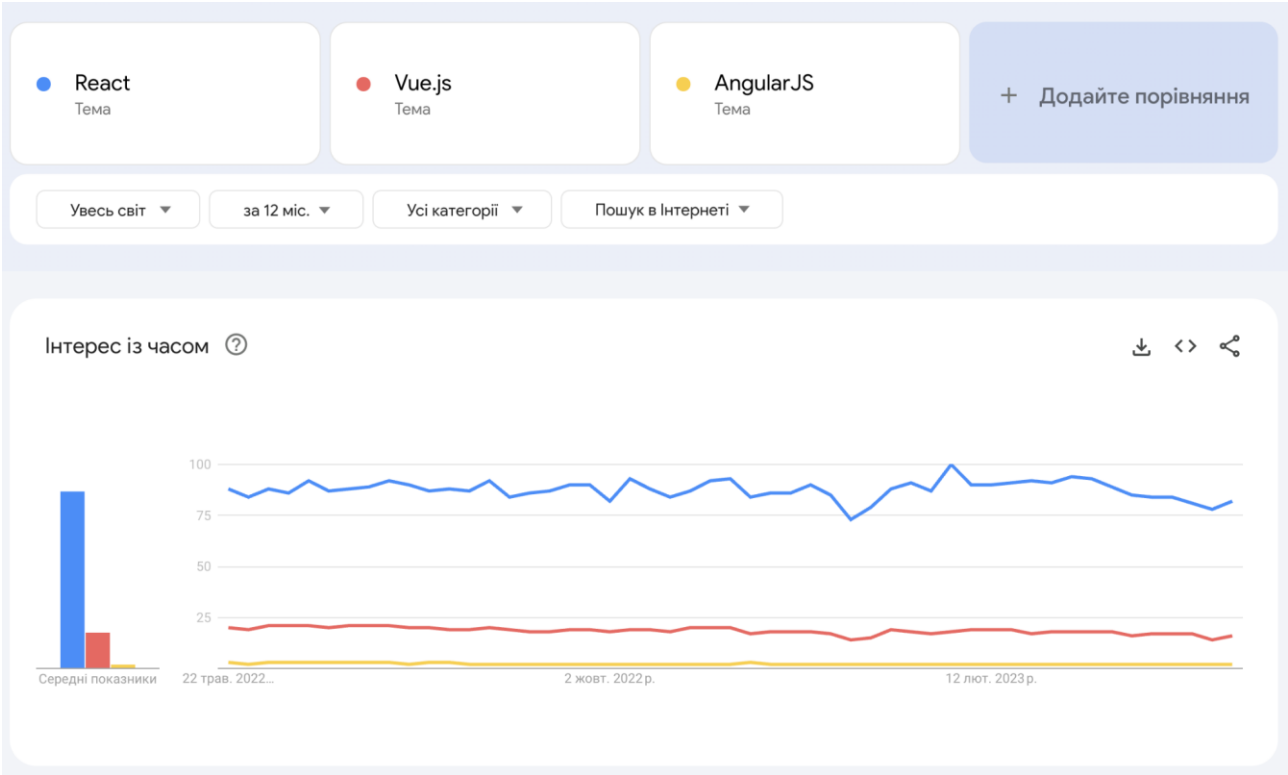

Рисунок 1.2 – Популярність запитів трьох фреймворків

Angular є повнофункціональним фреймворком, розробленим Google. Він надає потужні інструменти для розробки складних клієнтських додатків. Angular пропонує шаблони, залежність від ін'єкції, маршрутизацію та багато іншого. Приклади використання Angular включають Google, Microsoft, IBM.

Vue.js є легким і простим у використанні фреймворком, що набуває популярності. Він пропонує реактивну систему шаблонів та компонентну архітектуру, схожу на React.js. Vue.js також надає можливості для плавного оновлення компонентів і масштабування додатків. Деякі приклади використання Vue.js – Alibaba, Xiaomi, Adobe.

При виборі фреймворка для клієнтської частини додатку, важливо звернути увагу на кілька аспектів. Потрібно враховувати розмір фреймворка, продуктивність, наявність необхідних функцій, екосистему та підтримку спільноти розробників.

У випадку нашого додатку було вирішено використовувати React.js через його гнучкість, швидкість розробки та велику спільноту розробників. React.js дозволить нам створити компонентну структуру додатку, ефективно керувати

станом інтерфейсу та забезпечити швидке оновлення елементів. Крім того, React.js має широкий набір інструментів та бібліотек, які полегшують розробку та підтримку додатку.

2 СЕРВЕРНА ЧАСТИНА

2.1 Фреймворк для серверної частини

У цьому розділі ми зосередимось на фреймворку для серверної частини додатку – NestJS. Розробка серверної частини веб-додатків є надзвичайно важливою, оскільки вона забезпечує обробку запитів, взаємодію з базою даних та реалізацію бізнес-логіки.

NestJS – це прогресивний фреймворк для розробки серверних додатків на Node.js, який поєднує елементи об'єктно-орієнтованого та функціонального програмування [1]. Його основні принципи – модульність, масштабованість та ефективність. NestJS пропонує розробникам чистий і структурований код, шляхом використання концепцій, відомих з фреймворків, таких як Angular та Spring.

Головними причинами обрання NestJS для нашого проєкту є:

- TypeScript: NestJS повністю підтримує TypeScript, що дозволяє розробникам писати код з використанням сучасного статично типізованого JavaScript (це полегшує розробку, забезпечує підказки про типи та знижує кількість помилок під час виконання);
- модульна структура: NestJS надає модульну архітектуру, яка дозволяє логічно розділити додаток на незалежні модулі (це полегшує підтримку коду, розширення функціональності та тестування);
- ефективність: NestJS побудований на основі Node.js та використовує його потужні можливості неблокуючого вводу/виводу (non-blocking I/O) (це дозволяє створювати ефективні серверні додатки, що можуть обробляти багато запитів одночасно);
- розширюваність: завдяки модульній структурі та підтримці стандартних інструментів для роботи з базами даних, NestJS забезпечує

легку розширюваність додатку та інтеграцію з різними сторонніми бібліотеками та сервісами.

У наступному розділі ми розглянемо детальніше основні концепції та можливості, що пропонує NestJS.

2.2 Основні концепції та можливості фреймворку NestJS

NestJS пропонує ряд основних концепцій та можливостей, які допомагають розробникам писати чистий, масштабований та ефективний код для серверної частини додатків. Розглянемо детальніше про ці концепції та можливості.

Модулі. NestJS базується на модульній структурі, що дозволяє організувати код додатку у незалежні, повторно використовувані модулі. Модулі визначають контекст додатку та його функціональність. Це дозволяє розбити додаток на логічні компоненти та забезпечує легкість утримання та розширення.

Контролери. У NestJS контролери виконують роль обробників маршрутів із Express або контролерів із MVC-підходу. Вони відповідають на запити, оброблюють дані та повертають результати. Контролери дозволяють організовувати логіку роутингу та забезпечують чітке розділення між маршрутизацією та бізнес-логікою.

Провайдери. У NestJS провайдери відповідають за створення та управління екземплярами класів. Вони дозволяють впроваджувати ін'єкцію залежностей та створювати масштабовані компоненти додатку. Провайдери можуть бути використані для доступу до баз даних, сторонніх сервісів, кешування та інших функціональностей.

Сервіси. Сервіси у NestJS представляють бізнес-логіку додатку та виконують специфічні завдання. Вони використовуються контролерами для обробки запитів та взаємодії з провайдерами для отримання необхідних ресурсів. Сервіси допомагають створювати відокремлену логіку та полегшують

тестування.

Middleware. У NestJS можна використовувати middleware для обробки запитів перед тим, як вони досягнуть контролера. Middleware можна використовувати для автентифікації, авторизації, обробки помилок та інших завдань, що вимагають проміжного обробника.

Фільтри. Фільтри в NestJS використовуються для обробки виняткових ситуацій або валідації даних перед виконанням контролера або після його виконання. Вони дозволяють зробити централізовану обробку помилок та встановити правила валідації даних.

Пайпи. Пайпи використовуються для перевірки та обробки вхідних даних перед передачею їх до контролера або відповіді. Вони дозволяють здійснювати валідацію, преобразування та фільтрацію даних.

Ці основні концепції та можливості NestJS сприяють розробці масштабованих та легко утримуваних серверних додатків.

2.3 База даних

Справжнім стовпом будь-якого інформаційного проєкту є його база даних. Вона виступає в якості надійного сховища для даних, які потрібно зберігати, організовувати та маніпулювати. Розуміння основних концепцій баз даних та їх ефективного використання є критично важливим для успішної розробки будь-якого додатку.

База даних є центральним елементом будь-якої інформаційної системи, і вона відіграє ключову роль у зберіганні, організації та керуванні даними. Вона дозволяє нам ефективно зберігати, оновлювати, вилучати та аналізувати інформацію.

Перед тим, як розпочати цей розділ, важливо з'ясувати сутність Системи керування базами даних (СКБД). СКБД – це важлива складова сучасних

інформаційних систем, яка дозволяє створювати, керувати та маніпулювати базами даних. Вона надає механізми для зберігання, організації, пошуку та доступу до даних, а також забезпечує безпеку, цілісність та ефективну обробку інформації.

СКБД розрізняються за різними характеристиками, такими як тип бази даних, масштабованість, продуктивність, безпека та підтримувані функції. Розглянемо декілька прикладів існуючих СКБД.

PostgreSQL. Це одна з найпопулярніших відкритих реляційних СКБД. Вона володіє потужними можливостями, широкою підтримкою стандартів SQL і високою надійністю. PostgreSQL забезпечує широкий спектр функцій, включаючи транзакції, індексацію, реплікацію та підтримку географічних даних.

MySQL. Ця реляційна СКБД є однією з найпопулярніших у світі. Вона використовується для різноманітних застосувань, від невеликих веб-сайтів до великих корпоративних систем. MySQL володіє високою продуктивністю, широким спектром функцій і підтримкою різних програмних інтерфейсів.

MongoDB. Це документ-орієнтована NoSQL СКБД, яка забезпечує гнучкість та швидкодію. MongoDB дозволяє зберігати дані у вигляді документів у форматі JSON, що спрощує роботу з невструктурованими даними. Вона часто використовується для сучасних веб-додатків, аналітики даних та інших сфер, де потрібна горизонтальна масштабованість.

Microsoft SQL Server. Це комерційна реляційна СКБД від компанії Microsoft. Вона пропонує широкий спектр функцій для управління та аналізу даних, включаючи транзакції, зберігання процедур, аналітичні можливості та інтеграцію з іншими продуктами Microsoft.

Ці СКБД представляють різні типи і підходи до зберігання та керування даними. Вибір певної СКБД залежить від потреб проєкту, його масштабу, функціональних вимог та інших факторів, що потребують уважного вивчення та аналізу.

2.3.1 Реляційна модель

Реляційна модель є однією з найпоширеніших моделей баз даних і використовується для організації та управління даними. Вона ґрунтується на математичній теорії множин і логіці першого порядку та пропонує структурований підхід до зберігання та обробки даних. Основною ідеєю реляційної моделі є використання таблиць, відомих як «реляційні таблиці», для представлення даних та встановлення взаємозв'язків між ними [6].

Розглянемо основні поняття реляційної моделі.

Реляційна таблиця. Це основна структурна одиниця реляційної моделі. Вона представляє собою двовимірну сітку з рядками і стовпцями, де кожен рядок відповідає запису (кортежу), а кожний стовпець представляє атрибут (поле). Кожна клітина таблиці містить одне значення.

Ключ. Ключ є унікальним ідентифікатором для кожного запису в таблиці. Він дозволяє однозначно ідентифікувати кожний рядок і встановлювати зв'язки між різними таблицями.

Відношення. Відношення вказує на зв'язок між двома таблицями, який встановлюється за допомогою спільних атрибутів. Відношення може бути один до одного, один до багатьох або багато до багатьох [7].

Нормалізація. Це процес організації даних в таблицях таким чином, щоб уникнути дублікації і забезпечити цілісність та ефективну обробку даних. Нормалізація реляційних таблиць допомагає зменшити аномалії та підтримує структурну чистоту бази даних.

Реляційна модель має багато переваг, включаючи простоту у використанні, гнучкість, можливість ефективної обробки запитів та забезпечення цілісності даних. Вона широко використовується у багатьох сферах, включаючи бізнес, науку, освіту та багато інших.

2.3.2 Нереляційна модель

Нереляційна модель бази даних – це альтернативний підхід до зберігання та обробки даних, який відрізняється від традиційної реляційної моделі. Вона дозволяє зберігати дані у більш гнучкий та неструктурований спосіб, що дозволяє ефективно вирішувати певні завдання та сценарії.

Основні особливості нереляційних баз даних:

- неструктуровані дані: нереляційна модель дозволяє зберігати дані без жорсткої структури, що дозволяє працювати з неструктурованими або напівструктурованими даними, такими як JSON, XML, документи, зображення та інші формати;
- гнучкість схеми: у нереляційних базах даних немає жорсткої вимоги до попередньо визначеної схеми даних (це означає, що можна додавати, змінювати або видаляти поля в документах без необхідності виконання складних операцій зміни схеми);
- горизонтальне масштабування: нереляційні бази даних добре підходять для горизонтального масштабування, що означає розподіл даних на багато вузлів для забезпечення високої доступності та продуктивності;
- висока продуктивність: завдяки своїй структурі та оптимізаціям, нереляційні бази даних можуть забезпечувати високу продуктивність при обробці великого обсягу даних.

Деякі приклади нереляційних баз даних включають:

- MongoDB: MongoDB є популярною нереляційною базою даних, яка зберігає дані у вигляді JSON-подібних документів (вона широко використовується для зберігання напівструктурованих даних та масштабованих веб-додатків);
- Cassandra: Cassandra є розподіленою нереляційною базою даних, яка призначена для масштабованого зберігання великого обсягу даних на багатьох вузлах (вона часто використовується для забезпечення високої доступності та швидкодії при роботі з великими навантаженнями);

- Redis: Redis є ключ-значенням базою даних, яка забезпечує швидке зберігання та доступ до даних у пам'яті (вона часто використовується для кешування даних, сесійного зберігання та роботи з потоками даних);
- Neo4j: Neo4j є графовою базою даних, яка зберігає дані у вигляді вузлів та зв'язків (вона спеціалізується на зберіганні та обробці взаємозв'язаних даних, таких як соціальні мережі, рекомендації та графові алгоритми).

Ці приклади показують різноманітність нереляційних баз даних та їхню використовуваність для різних типів даних та сценаріїв. Вибір нереляційної бази даних залежить від специфіки проєкту та вимог до зберігання й обробки даних

2.3.3 Порівняння реляційної та нереляційної моделей

Реляційна модель і нереляційна модель є двома різними підходами до організації та зберігання даних. Основна різниця між ними полягає в структурі та способі представлення даних.

Давайте детальніше розглянемо цю різницю.

Структура даних:

- реляційна модель базується на табличній структурі, де дані зберігаються у вигляді таблиць з рядками і стовпцями (у цій моделі використовуються ключі, зв'язки та обмеження цілісності для організації даних);
- нереляційна модель дозволяє зберігати дані у більш гнучкому форматі, такому як документи, графи або ключ-значення (вона не має жорсткої вимоги до структури даних, що дозволяє зберігати неструктуровані або напівструктуровані дані).

Складність схеми:

- в реляційній моделі необхідно попередньо визначити схему даних,

тобто визначити таблиці, поля, типи даних та зв'язки між ними (зміни в схемі можуть бути складними і вимагати великої кількості зусиль та часу);

– нереляційна модель не має жорсткої вимоги до схеми даних, що дозволяє більшу гнучкість (можна додавати, змінювати або видаляти поля без необхідності виконання складних операцій зміни схеми).

Масштабованість:

– реляційні бази даних часто мають складну архітектуру, що може бути обмеженою при масштабуванні горизонтально або вертикально (великі обсяги даних можуть вимагати складних операцій реплікації та шарування);
– нереляційні бази даних, зокрема графові та колоночні, зазвичай мають легшу масштабованість (вони можуть бути легко розподілені на кластери або шари для обробки великих обсягів даних та високої доступності).

Запити та операції:

– реляційна модель має стандартну мову запитів SQL, яка дозволяє виконувати складні операції над даними, такі як з'єднання, групування, фільтрація тощо (вона має потужний набір операцій для маніпулювання даними);
– нереляційні бази даних мають свої власні механізми запитів, які можуть бути специфічними для типу бази даних (наприклад, графові бази даних мають запити, спрямовані на роботу зі зв'язками між вузлами, а ключ-значення бази даних підтримують прості операції зчитування та запису).

Використання даних:

– реляційні бази даних зазвичай використовуються для ситуацій, де важлива структурованість даних, забезпечення цілісності та виконання складних запитів (вони часто використовуються в бізнес-середовищах, де потрібна точність та надійність даних);

– нереляційні бази даних використовуються в ситуаціях, коли дані можуть бути неструктурованими, великими за обсягом або змінюватися динамічно (вони зручні для веб-додатків, аналітики даних, соціальних мереж, де потрібна гнучкість та швидкодія).

Це лише деякі основні різниці між реляційною та нереляційною моделями. Вибір між ними залежить від потреб, характеру даних та вимог проєкту. Важливо враховувати специфіку завдання та аналізувати переваги та недоліки кожної моделі перед прийняттям рішення.

2.3.4 Вибір СКБД

У даному розділі ми розглянемо процес вибору СКБД і визначимо ключові аспекти, які слід враховувати при цьому. Вибір відповідної СКБД є важливим завданням, оскільки це впливає на продуктивність, надійність і можливості бази даних.

Одним з перших кроків у виборі СКБД є з'ясування вимог і особливостей проєкту. Враховувати потреби щодо швидкодії, масштабованості, надійності, безпеки та зручності роботи з базою даних. Для цього потрібно провести аналіз вимог, дослідити ринок СКБД та оцінити їх можливості.

Крім того, слід враховувати тип даних, які будуть зберігатися у базі даних, обсяг даних, частоту змін та необхідність складних операцій обробки даних. Вибір СКБД повинен враховувати також бюджет проєкту та наявність кваліфікованого персоналу для роботи з обраною СКБД. Важливо мати чітке розуміння переваг і недоліків кожного типу СКБД для того, щоб здійснити обґрунтований вибір, відповідний конкретним потребам проєкту.

Правильно підібрана СКБД дозволить забезпечити надійну, швидку та ефективну роботу з базою даних, що в свою чергу сприятиме успішному виконанню поставлених завдань та досягненню поставлених цілей проєкту.

Тож, розглянемо деякі найпопулярніші СКБД та опишемо їх недоліки та переваги.

MySQL є однією з найпоширеніших відкритих реляційних СКБД, яка володіє широким спектром застосувань. Вона відома своєю швидкодією, простотою в використанні та надійністю. MySQL має велику спільноту користувачів та розробників, що забезпечує багато ресурсів для підтримки та розвитку. Вона підтримує масштабування та реплікацію даних для обробки великих обсягів інформації. Однак, MySQL має обмежену підтримку деяких розширених функцій та може мати обмеження в продуктивності під великим навантаженням.

PostgreSQL є потужною та розширюваною відкритою реляційною СКБД. Вона володіє високою надійністю, масштабованістю та підтримує розширені функції. PostgreSQL має вбудовану підтримку розподіленого оброблення, що дозволяє працювати з розподіленими системами баз даних. Вона також надає широкі можливості керування даними, включаючи підтримку географічних та геолокаційних даних [3]. Однак, PostgreSQL може вимагати більшої конфігурації та ресурсів порівняно з іншими СКБД.

SQLite є легковагою, вбудовуваною реляційною СКБД, яка відома своєю простотою в використанні та низькими вимогами до ресурсів. Вона є однофайловою базою даних, що дозволяє зберігати всю інформацію в одному файлі без необхідності налаштування окремого сервера баз даних. SQLite підтримує стандартну SQL-синтаксис та має набір основних функцій для роботи з даними. Вона широко використовується в мобільних додатках, веб-браузерах та інших вбудованих системах. Однак, SQLite може бути менш потужною порівняно зі великими реляційними СКБД, особливо при роботі з великими обсягами даних або високим навантаженням.

2.4 API

API (інтерфейс програмування додатків) визначає набір правил, протоколів та процедур, які дозволяють різним компонентам програмного забезпечення взаємодіяти один з одним, обмінюватися даними та виконувати

певні функції. В контексті розробки веб-додатків та систем, API стає невід'ємною складовою, що забезпечує комунікацію між клієнтською та серверною частинами.

API може бути реалізований у різних форматах та протоколах, і його використання є основою для спільної роботи різних додатків, систем та сервісів. Завдяки API, розробники можуть розширювати функціональність своїх додатків, використовуючи функції та дані, які надаються іншими системами. Це відкриває безліч можливостей для інтеграції, розширення та покращення функціональності програмного забезпечення.

2.4.1 REST API

REST API (Representational State Transfer Application Programming Interface) є архітектурним стилем для розробки веб-сервісів, який базується на принципах REST [10]. Він надає спосіб взаємодії між клієнтськими додатками та серверами через мережу Інтернет.

REST API використовується для передачі даних та виконання операцій над цими даними через стандартні HTTP методи, такі як GET, POST, PUT та DELETE. Ці методи відповідають конкретним діям, які можна виконати з ресурсами, наприклад, отримати дані, створити новий запис, оновити існуючий запис або видалити його.

Розглянемо основні принципи REST API.

Ресурси. REST API моделює систему як набір ресурсів, які можуть бути доступні для клієнтів. Кожен ресурс має свій унікальний ідентифікатор (URI) та може бути доступний за допомогою стандартних HTTP методів.

Уніфікований інтерфейс. REST API використовує єдиний набір принципів та стандартів для взаємодії з ресурсами. Це включає використання HTTP протоколу, стандартних методів, кодів стану HTTP і форматів передачі даних, таких як JSON або XML.

Безстанність. REST API не зберігає жодної інформації про стан клієнта на сервері між запитами. Кожен запит містить всю необхідну інформацію для його обробки, що забезпечує простоту та масштабованість системи.

Кешування. REST API підтримує можливість кешування відповідей сервера клієнтами. Це дозволяє знизити навантаження на сервер і покращити продуктивність системи.

REST API має кілька переваг, включаючи простоту використання, масштабованість, незалежність від платформи та можливість використання стандартних протоколів та інструментів. Він забезпечує стандартизований спосіб комунікації між різними системами та дозволяє розробникам створювати розширювані та легкозмінні інтерфейси.

Узагальнюючи, REST API є потужним засобом для взаємодії клієнтських додатків та серверів у розподілених системах. Він забезпечує стандартизований підхід до розробки веб-сервісів, що сприяє швидкому та ефективному обміну даними між різними системами.

2.4.2 RESTful API

RESTful API (Representational State Transfer) є одним з найпоширеніших стандартів для проєктування та реалізації веб-сервісів, який базується на архітектурному стилі REST. Він використовується для забезпечення ефективної комунікації між клієнтськими додатками та серверами через Інтернет.

RESTful API розглядає ресурси (наприклад, дані або функціональні можливості) як основні елементи системи. Він використовує HTTP протокол для виконання операцій над цими ресурсами за допомогою чотирьох основних методів: GET (отримати), POST (створити), PUT (оновити) і DELETE (видалити). Кожен метод відповідає конкретній дії, яку клієнтський додаток бажає виконати з ресурсом.

Розглянемо основні принципи RESTful API.

Архітектура клієнт-сервер (Client-server architecture). RESTful API розділяє клієнтську та серверну частини системи, що дозволяє їм розвиватись незалежно одне від одного. Це забезпечує модульність, масштабованість та розділення відповідальностей між компонентами системи.

Відсутність стану (Statelessness). RESTful API не зберігає жодної інформації про стан клієнта на сервері між запитами. Кожен запит містить всю необхідну інформацію для його обробки, що забезпечує простоту та масштабованість системи.

Однорідний інтерфейс (Uniform interface). RESTful API використовує єдиний та уніфікований набір принципів для взаємодії з ресурсами. Це включає використання URI (ідентифікатори ресурсів), стандартних HTTP методів, форматів передачі даних, таких як JSON або XML, та використання кодів стану HTTP для передачі результатів запитів.

RESTful API забезпечує кілька переваг, включаючи простоту використання, широку підтримку, незалежність від платформи та масштабованість. Він є стандартом веб-розробки, який використовується багатьма великими компаніями та організаціями для розробки розподілених систем та веб-сервісів.

Завершуючи, RESTful API є потужним та ефективним інструментом для забезпечення комунікації між клієнтськими додатками та серверами у розподілених системах. Він дозволяє розробникам створювати гнучкі та легкозмінні інтерфейси, що дозволяють взаємодіяти з ресурсами системи шляхом стандартних HTTP запитів. Цей підхід сприяє створенню масштабованих та розширюваних програмних рішень, що відповідають сучасним потребам веб-розробки.

3 ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ

Проєктування веб-додатку є важливим етапом в розробці програмного забезпечення, який включає в себе визначення архітектури, структури даних, інтерфейсу користувача та інших ключових аспектів системи. В цьому розділі будуть розглянуті принципи та методи проєктування веб-додатків, які допоможуть забезпечити ефективність, масштабованість та надійність розроблюваної системи.

3.1 Вимоги до системи

Першим кроком у проєктуванні веб-додатку є аналіз та визначення вимог до системи. Це включає в себе збір та аналіз потреб користувачів, визначення функціональних та нефункціональних вимог, а також визначення обмежень та умов розробки. Важливо враховувати потреби користувачів та бізнес-вимоги, щоб забезпечити успішне виконання проєкту.

Вимоги до системи відіграють ключову роль у проєктуванні веб-додатку, оскільки вони визначають основні потреби користувачів та бізнес-вимоги, які мають бути задоволені системою. Аналіз та визначення вимог є першим етапом проєктування і забезпечує правильне спрямування розробки та успішне виконання проєкту.

3.1.1 Функціональні вимоги

Функціональні вимоги визначають функціональність системи, тобто ті дії та операції, які система повинна здійснювати для виконання своїх завдань. Вони описують функції, які мають бути доступні для користувачів та можуть

включати такі елементи, як реєстрація користувачів, авторизація, створення та редагування даних, взаємодія з іншими системами тощо. Наприклад, вимога може бути сформульована так: «Система повинна забезпечувати можливість створення нового користувача з обов'язковим заповненням полів: ім'я, електронна пошта, пароль».

3.1.2 Нефункціональні вимоги

Нефункціональні вимоги визначають характеристики та якості системи, які не пов'язані безпосередньо з її функціональністю. Ці вимоги описують властивості системи, які впливають на її продуктивність, безпеку, масштабованість, надійність та інші аспекти. Наприклад, нефункціональна вимога може звучати так: «Система повинна забезпечувати швидкий час відгуку не більше ніж 2 секунди для кожного запиту користувача».

3.1.3 Обмеження та умови розробки

Обмеження та умови розробки визначають обмеження, які необхідно враховувати під час проєктування та розробки системи. Це можуть бути технічні обмеження, такі як підтримка певних мов програмування чи фреймворків, обмеження на доступні ресурси (пам'ять, пропускна здатність мережі) тощо. Також можуть встановлюватись умови щодо термінів розробки, бюджету, команди розробників та інших факторів, які можуть вплинути на процес створення системи.

Вимоги до системи є важливим кроком у проєктуванні веб-додатку, оскільки вони становлять основу для подальшого аналізу, проєктування та розробки. Ці вимоги повинні бути вичерпними, зрозумілими та детально сформульованими, щоб уникнути непорозумінь та забезпечити успішне втілення поставлених цілей.

3.2 Архітектура системи

Архітектура системи визначає структуру, організацію та взаємозв'язки між компонентами системи. Вона визначає, як система буде розгортатися, розвиватися, масштабуватися та підтримуватися на протязі її життєвого циклу. Архітектура веб-додатку включає в себе прийняття рішень щодо розміщення функціональності, взаємодії компонентів, забезпечення безпеки, швидкодії та розширюваності системи.

3.2.1 Монолітна архітектура

Монолітна архітектура є однією з найпоширеніших архітектурних моделей для веб-додатків. У цій моделі весь додаток складається з одного компонента, який зазвичай є монолітним за своєю структурою. Це означає, що всі логічні компоненти, такі як користувацький інтерфейс, бізнес-логіка та доступ до бази даних, об'єднані в одному програмному модулі.

Основними особливостями монолітної архітектури є:

- централізована логіка: усі компоненти додатку розташовані в одному монолітному модулі, що спрощує розробку та розгортання додатку;
- простота розробки: завдяки тому, що всі компоненти належать до одного модуля, команда розробників може легко спілкуватися та співпрацювати під час розробки;
- простота тестування: тести можуть бути легко написані та виконані на монолітній архітектурі, оскільки всі компоненти знаходяться в одному контексті;
- масштабованість: монолітна архітектура може бути масштабована горизонтально, шляхом додавання додаткових інстансів моноліта для розподілення навантаження.

Незважаючи на переваги, монолітна архітектура також має свої недоліки:

- складність підтримки: у монолітних додатках складніше внести зміни, оскільки всі компоненти залежать один від одного;
- обмежена масштабованість: монолітні додатки можуть стикатися з обмеженнями щодо масштабування, оскільки вся логіка розташована в одному модулі;
- висока залежність: зміни в одному компоненті можуть впливати на інші, що може призвести до непередбачуваних наслідків;
- обмежена технологічна гнучкість: монолітна архітектура може бути обмежена в виборі технологій, оскільки всі компоненти повинні працювати разом в одному контексті.

Хоча монолітна архітектура має свої обмеження, вона є ефективним варіантом для невеликих та середніх веб-додатків, які не вимагають високої масштабованості та технологічної гнучкості. Розробка та розгортання монолітних додатків є більш простою та дешевою, що робить їх популярними серед підприємств з обмеженими ресурсами [8]. На рисунку 3.1 зображено структурна схему монолітної архітектури.

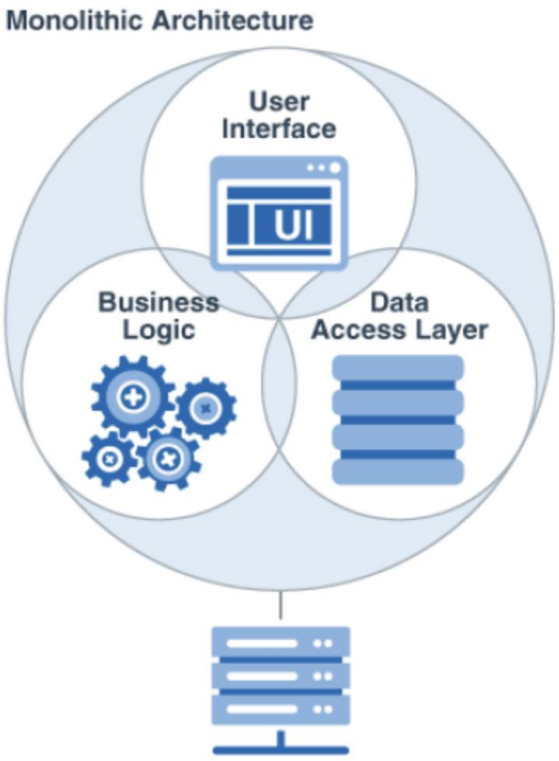

Рисунок 3.1 – Структурна схема монолітної архітектури

3.2.2 Модель-представлення-контролер (MVC) архітектура

Модель-представлення-контролер (MVC) є однією з найпоширеніших архітектурних моделей для веб-додатків. Вона розділяє логіку програми на три основні компоненти: модель, представлення та контролер.

Модель відповідає за представлення даних та бізнес-логіку додатку. Вона забезпечує доступ до даних, валідацію, обробку бізнес-правил та виконання операцій з базою даних. Модель представляє собою абстракцію даних із зовнішніми системами та управляє всіма операціями, пов'язаними з даними.

Представлення відповідає за візуальне відображення даних та інтерфейс користувача. Воно відповідає за генерацію веб-сторінок, відображення даних, обробку подій користувача та взаємодію з користувачем. Представлення може бути реалізоване у вигляді HTML-шаблонів, CSS-стилів, JavaScript-коду та інших веб-технологій.

Контролер є посередником між моделлю та представленням. Він обробляє вхідні дані від користувача, викликає відповідні операції моделі, обробляє логіку додатку та встановлює зв'язок між моделлю та представленням. Контролер приймає запити від користувача, виконує необхідні операції та повертає відповідь у вигляді оновленого представлення.

MVC архітектура надає розділення відповідальностей між компонентами, що спрощує розробку, тестування та підтримку веб-додатків. Вона дозволяє легко змінювати логіку додатку, дизайн і взаємодію з користувачем незалежно один від одного. Крім того, MVC сприяє повторному використанню компонентів, полегшує роботу команди розробників та забезпечує масштабованість додатків.

Обрання MVC архітектури для нашого веб-додатку було обґрунтовано наступними перевагами:

- логічна структура: MVC дозволяє явно розділити логічні компоненти додатку, що сприяє покращенню читабельності, розширюваності та обліку внесених змін;

- модульність: компоненти можуть бути розроблені та підтримуватись окремо, що сприяє швидкому впровадженню змін та розвитку додатку;
- тестування: модульність дозволяє проводити ефективне тестування окремих компонентів, забезпечуючи високу якість та стабільність системи;
- гнучкість: застосування MVC дозволяє легко змінювати та модифікувати компоненти системи без впливу на інші частини;

Враховуючи зазначені переваги, MVC архітектура відповідає вимогам нашого веб-додатку та забезпечує його ефективну розробку, підтримку та розширення.

На рисунку 3.2 зображено структурна схему MVC архітектури.

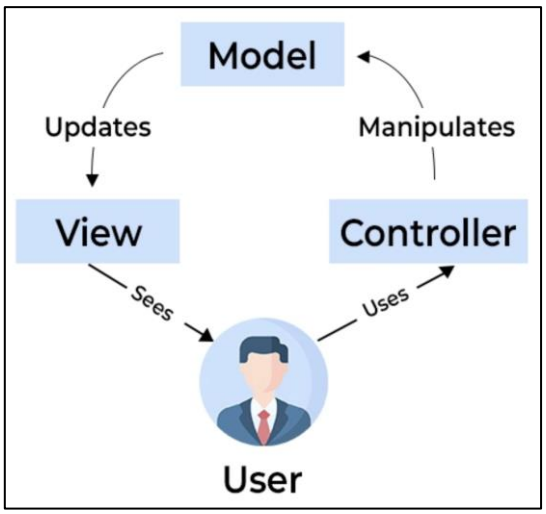

Рисунок 3.2 – Структурна схема MVC архітектури

3.2.3 Мікросервісна архітектура

Мікросервісна архітектура є однією з популярних моделей архітектури програмного забезпечення, яка полягає в розбитті додатку на набір незалежних мікросервісів [2]. Кожен мікросервіс є окремим функціональним модулем, який виконує обмежений набір функцій та взаємодіє з іншими мікросервісами через механізм API.

Дизайн мікросервісної архітектури базується на принципі однієї відповідальності (Single Responsibility Principle), де кожен сервіс відповідає за свою конкретну функцію [4]. Це дозволяє забезпечити високу гнучкість, швидкість розробки та розгортання, а також полегшує масштабування окремих компонентів системи.

На рисунку 3.3 зображено структурну схему мікросервісної архітектури.

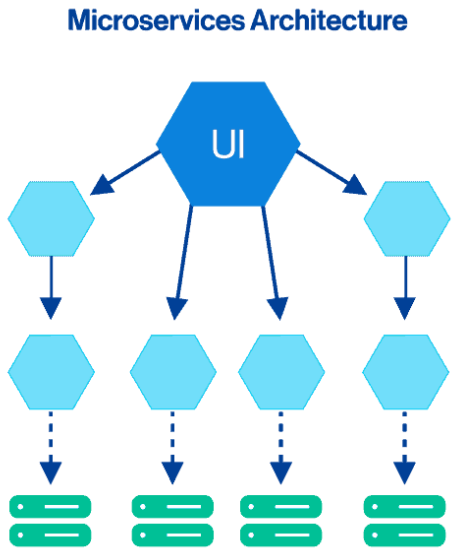

Рисунок 3.3 – Структурна схема мікросервісної архітектури

Основні принципи мікросервісної архітектури включають:

– розбиття на сервіси: додаток розбивається на незалежні сервіси, кожен з яких виконує конкретну функцію (кожен сервіс може бути розроблений, розгорнутий та масштабований окремо);
– комунікація через API: мікросервіси взаємодіють між собою через стандартизовані API, такі як HTTP або Message Queue (це дозволяє забезпечити локалізацію змін та незалежність сервісів);
– розгортання та масштабування: кожен мікросервіс може бути розгорнутий та масштабований окремо, що дозволяє ефективне використання ресурсів та гнучке масштабування в залежності від навантаження;

- легка заміна та розширення: завдяки розділенню на незалежні сервіси, заміна або розширення конкретного сервісу стає простим та мінімально впливає на інші компоненти системи [9];
- самостійна база даних: кожен мікросервіс може мати власну базу даних, що дозволяє краще керування даними та забезпечує високу гнучкість.

3.2.4 Діаграма прецедентів

У даному розділі буде представлена діаграма прецедентів, яка є важливим інструментом в аналізі та проєктуванні системи. Діаграма прецедентів є частиною методології UML (Unified Modeling Language) і дозволяє моделювати функціональність системи з точки зору її користувачів.

В аналізі системи діаграма прецедентів використовується для ідентифікації основних акторів системи та їх взаємодії з системою через прецеденти. Актори – це ролі, які виконують користувачі системи, будь-то реальні особи, інші програмні системи або навіть зовнішні об'єкти. Прецеденти визначають конкретні дії, які актори можуть виконувати в системі.

Діаграма прецедентів складається з акторів, прецедентів та взаємозв'язків між ними. Актори зображуються у вигляді піктограм людей або інших сутностей, які взаємодіють з системою. Прецеденти зображуються у вигляді овалів і названі відповідно до дій, які вони представляють. Взаємозв'язки між акторами та прецедентами відображаються у вигляді стрілок, що вказують на напрямок взаємодії.

Діаграма прецедентів дозволяє визначити основні функціональні можливості системи та взаємодію зовнішніх акторів з системою. Вона сприяє зрозумінню потреб та очікувань користувачів, а також створенню бази для подальшого проєктування системи, включаючи інші види діаграм, такі як діаграма послідовностей та діаграма класів.

За допомогою діаграми прецедентів можна виявити ключові функціональність системи, ідентифікувати ролі користувачів та їх потреби, а також встановити основу для подальшої деталізації та розробки системи. Вона створює можливість для взаємодії з зацікавленими сторонами та забезпечує спільне розуміння функціональності системи між розробниками, замовниками та іншими учасниками проєкту.

Заключаючи, діаграма прецедентів є важливим інструментом в аналізі та проєктуванні системи. Вона дозволяє визначити функціональність системи з точки зору користувачів, ідентифікувати ролі та взаємодію зовнішніх акторів, а також створює основу для подальшої розробки та впровадження системи з урахуванням потреб та очікувань користувачів.

На рисунку 3.4 зображена діаграма прецедентів з двома акторами.

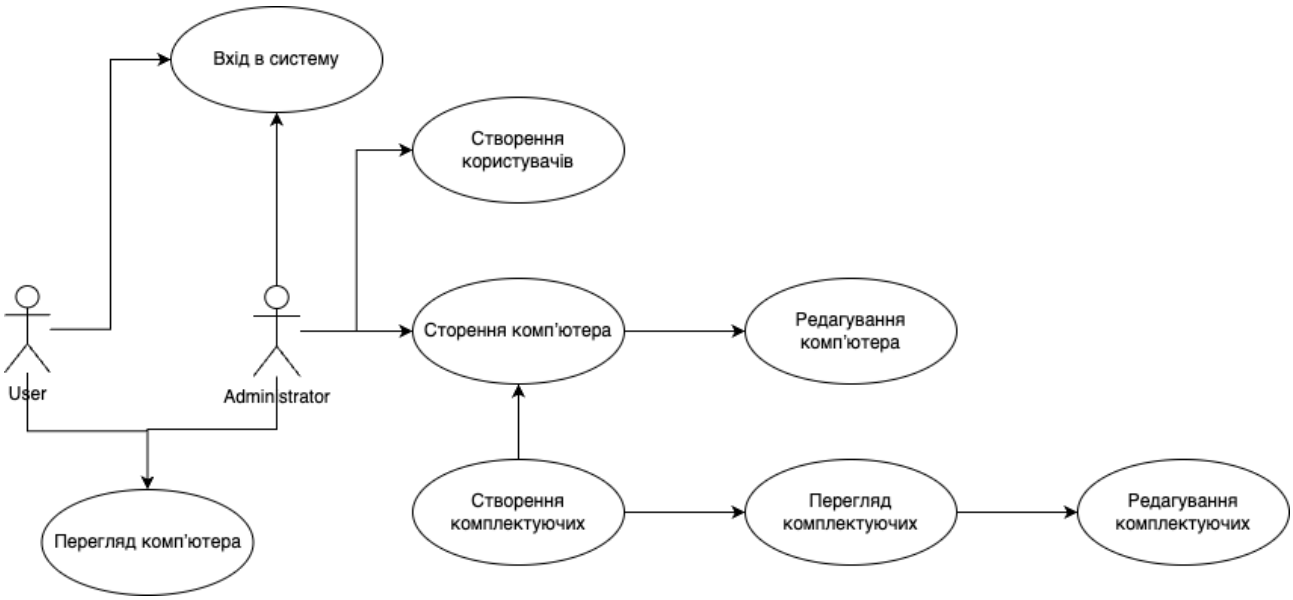

Рисунок 3.4 – Діаграма прецедентів

На діаграмі прецедентів зображено два актори адміністратор та користувач. Користувач може взаємодіяти тільки з комп’ютерами які належать йому як відповідальній особі. Адміністратор може взаємодіяти з будь-якими комп’ютерами. Він може створювати, оновлювати або видаляти комп’ютери з системи. Також, адміністратор може: створювати, оновлювати та видаляти

комплектуючі; призначати комп’ютерам користувачів як відповідальних за них осіб.

3.2.5 Концептуальне проєктування

Модель «сутність-взаємозв'язок» (Entity-Relationship або ER-модель) є широко використовуваним інструментом для концептуального проєктування баз даних. Ця модель дозволяє відображати сутності (entities), атрибути (attributes) та взаємозв'язки (relationships) між ними. Вона базується на концепції сутностей, які представляють реальні об'єкти або концепції, про які збирається інформація, а також на зв'язках між цими сутностями.

ER-модель надає графічний спосіб відображення структури бази даних. Кожна сутність має свій набір атрибутів, які описують її характеристики. Взаємозв'язки вказують на залежності між сутностями та їх тип, такі як один до одного, один до багатьох або багато до багатьох.

ER-модель дозволяє відображати структуру даних і зв'язки між ними, що допомагає розробникам краще розуміти вимоги до системи та визначати потрібні таблиці та їх зв'язки. Вона є важливим етапом проєктування бази даних і сприяє вдалому розподілу даних та забезпеченню ефективної роботи з ними

Перед створенням ER-діаграми, потрібно виділити основні сутності проєкту. Треба почати з сутностей комп’ютер і комплектуючі. Ці сутністі є фундаментальними для системи. Сутність computers має: первинний ключ id, status, inventoryNumber та відношення один-до-одного userId. Сутність комплектуючі має: первинний ключ id, title, description, відношення один-до-одного categoryId та image.

Необхідним також зробити окрему сутність для категоризації комплектуючих яка зветься component_categories, яка складається з первинного ключа id, name та відношення один-до-одного parentId.

На рисунку 3.5 зображена ER-діаграма до нашого застосунку.

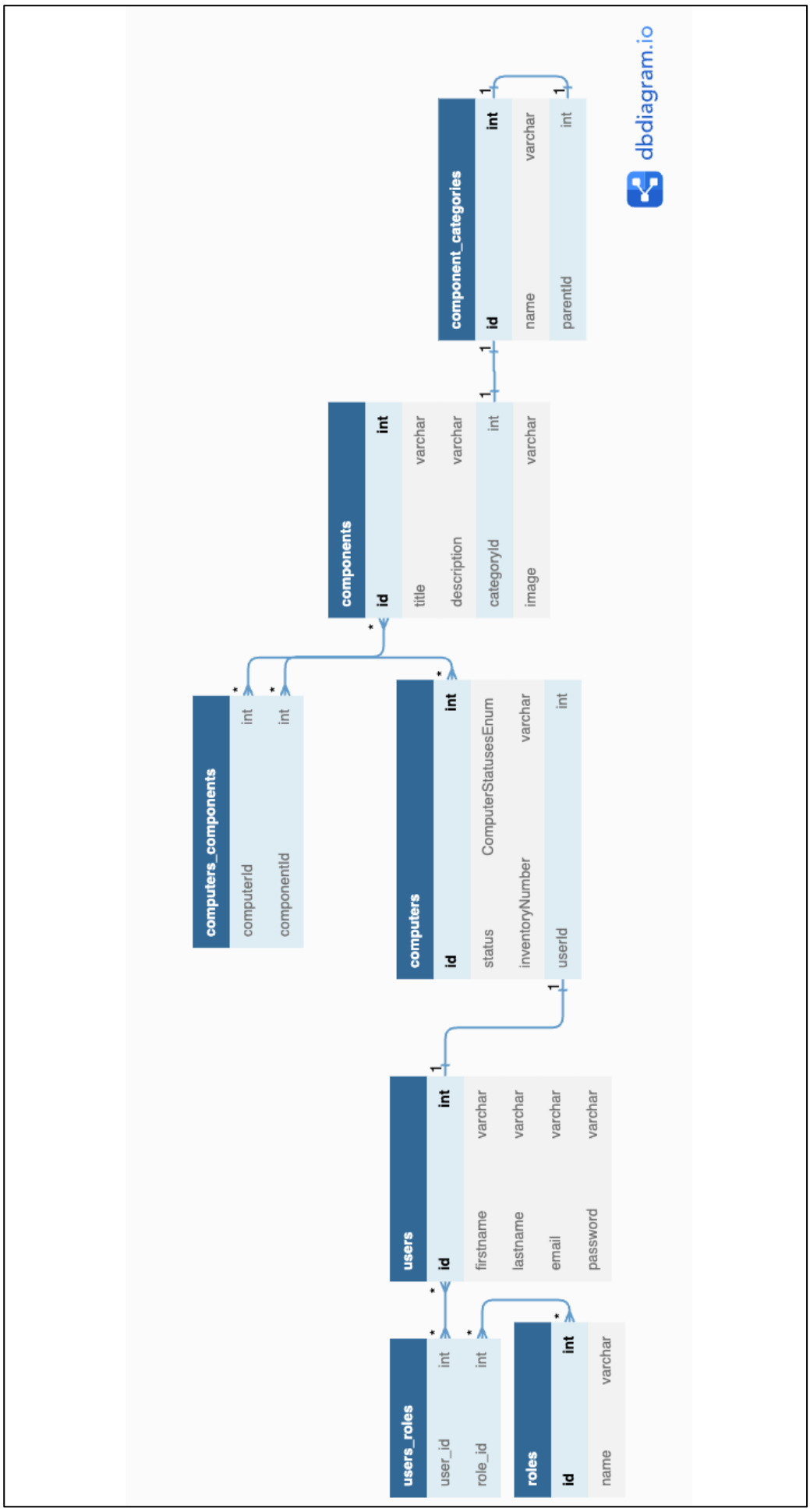

Рисунок 3.5 – ER-діаграма

Також потрібно зробити окрему сутність для користувачів та їх ролей. Сутність users буде складатися з первинного ключа id, firstname, lastname, email та password, а сутність roles буде складатися з первинного ключа id та name. Ці сутності необхідно об’єднати за допомогою сутності users_roles, яка складається з відношення багато-до-багатьох user_id та відношення багато-до-багатьох role_id.

3.2.6 Діаграма класів

У розробці програмного забезпечення діаграма класів є одним з основних інструментів для моделювання структури системи та відображення взаємозв'язків між класами. Вона надає графічний підхід до представлення класів, їх атрибутів та методів, а також спадкування та асоціацій між класами.

Діаграма класів складається з набору класів, які представляють об'єкти або концепції в системі. Кожен клас має свої атрибути, які описують його характеристики, та методи, які визначають його поведінку.

Діаграма класів дозволяє розробникам краще розуміти структуру системи, взаємозв'язки між класами та їхні властивості. Вона допомагає визначити потребу в класах, атрибутах та методах, а також з'ясувати, як класи взаємодіють один з одним. Діаграма класів також є важливим джерелом інформації для подальшого програмування та розробки системи, оскільки вона надає структурний огляд системи та розкриває її основні компоненти.

На рисунку 3.6 зображено діаграму класів до нашого додатку.

Загалом, діаграма класів є потужним інструментом для моделювання структури системи і відображення взаємозв'язків між класами. Вона допомагає зрозуміти систему на більш глибокому рівні та служить основою для подальшого проєктування та розробки програмного забезпечення.

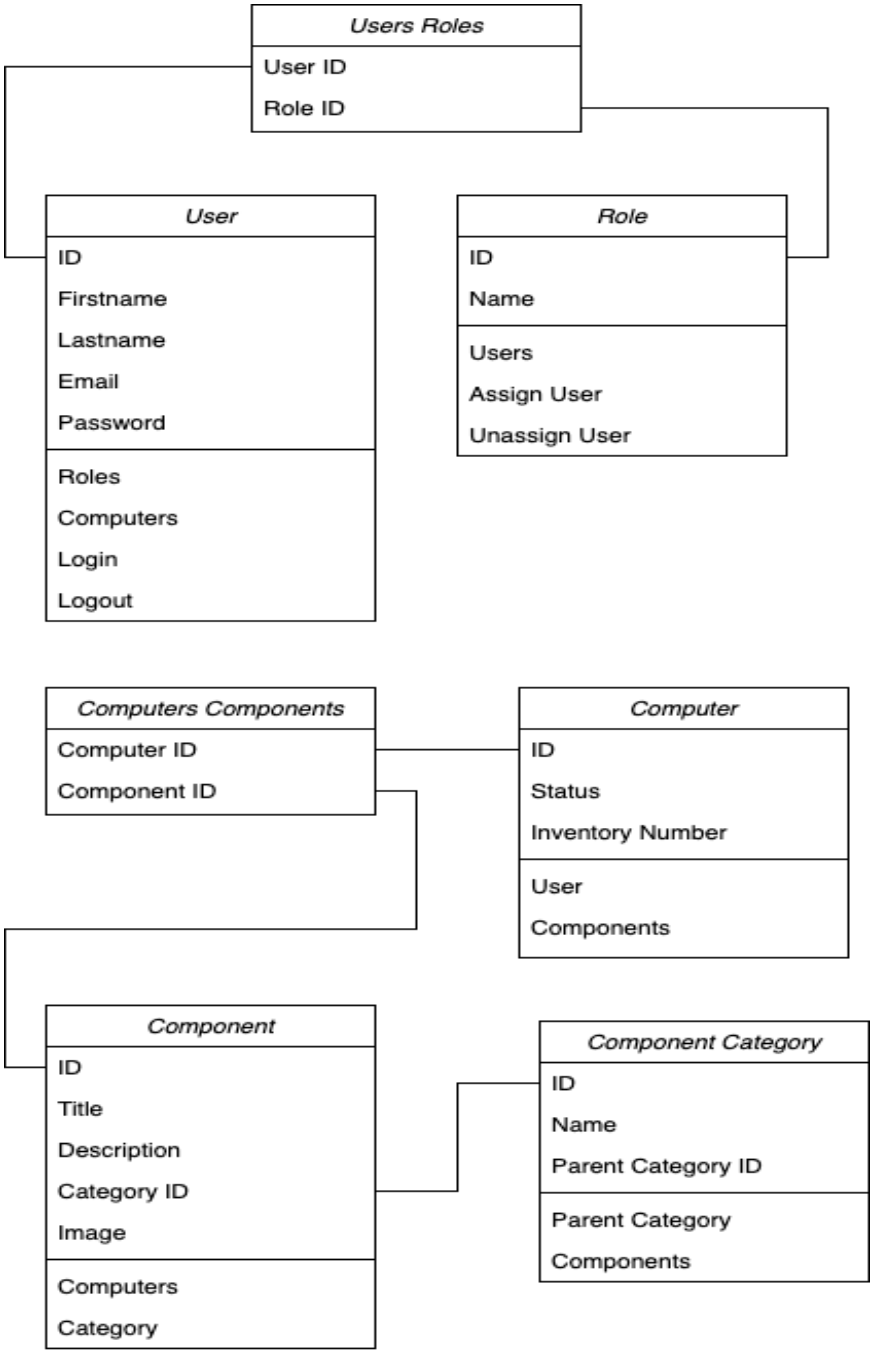

Рисунок 3.6 – Діаграма класів

4 ПРИКЛАД РОБОТИ ВЕБ-ДОДАТКА

У цьому розділі наведено скріншоти роботи веб-додатка.

Перше що побачить користувач увійшовши на сайт, це сторінка авторизації (див. рис. 4.1).

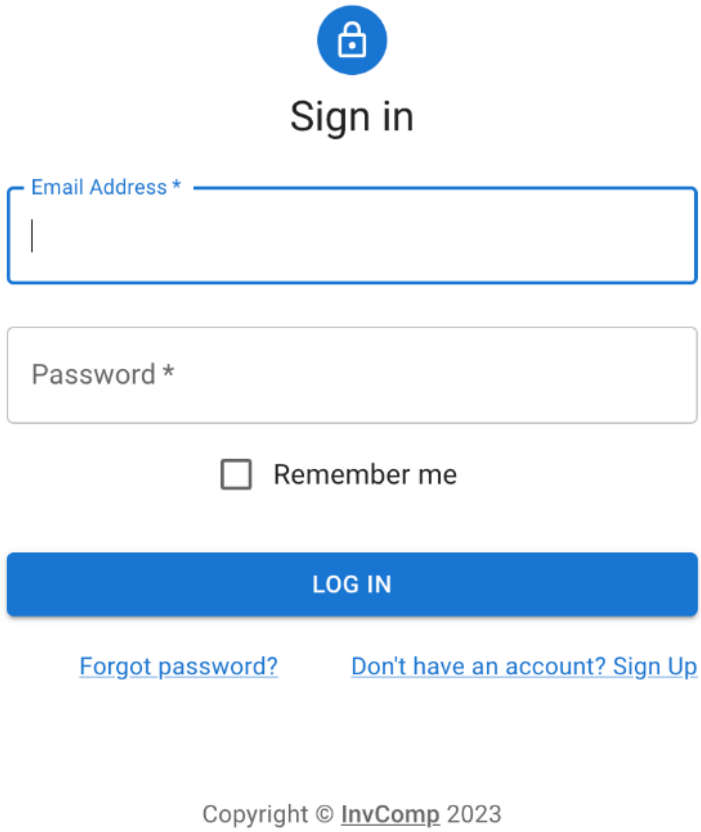

Рисунок 4.1 – Сторінка авторизації

Після успішної авторизації користувач побачить інформаційну панель (див. рис. 4.2). Якщо користувач має роль “Administrator”, то він побачить сторінку із користувачами та наступні пункти у меню зліва: Users, Roles, Computers, Components; інакше, користувач побачить лише список комп’ютерів, а в меню йому буде доступний лише пункт Computers.

На цій сторінці користувач (який має роль “Адміністратор”) може додавати або видаляти користувачів.

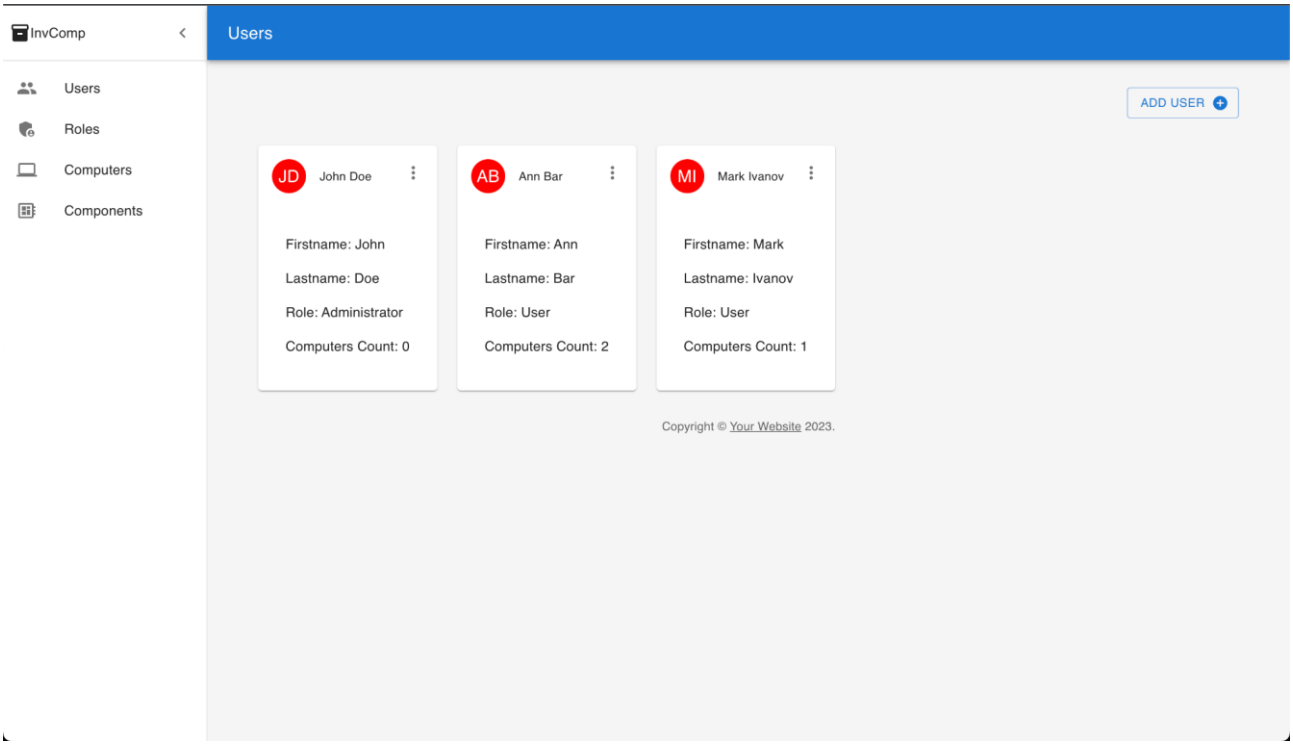

Рисунок 4.2 – Інформаційна панель на сторінці “Users”

Натиснувши на кнопку “Roles” в меню зліва, користувач перейде до сторінки на якій відображено список створених ролей (див. рис. 4.3). На цій сторінці адміністратор може створювати або видаляти ролі.

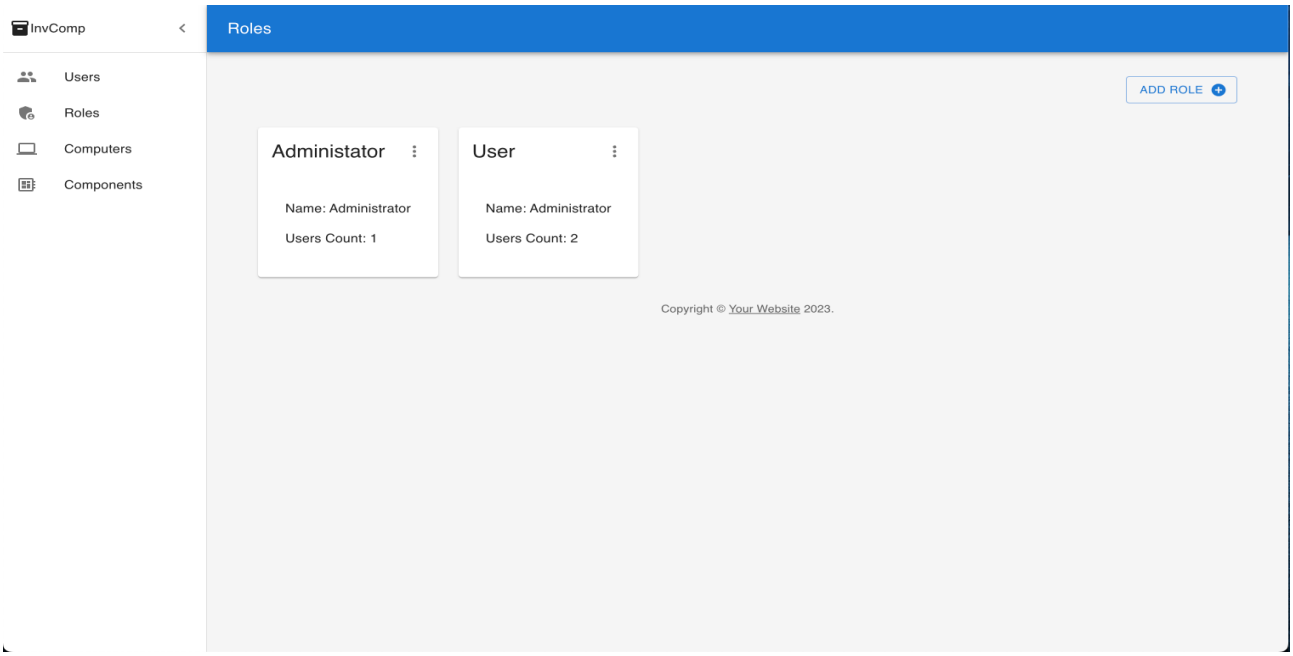

Рисунок 4.3 – Інформаційна панель на сторінці “Roles”

Натиснувши на кнопку “Computers” в меню зліва, користувач перейдо до сторінки, на якій відображено список комп’ютерів (див. рис. 4.4). На цій сторінці звичайний користувач (який має роль “User”) може лише переглядати інформацію про комп’ютери, а користувач який має роль “Administrator” може додавати, редагувати та видаляти комп’ютери.

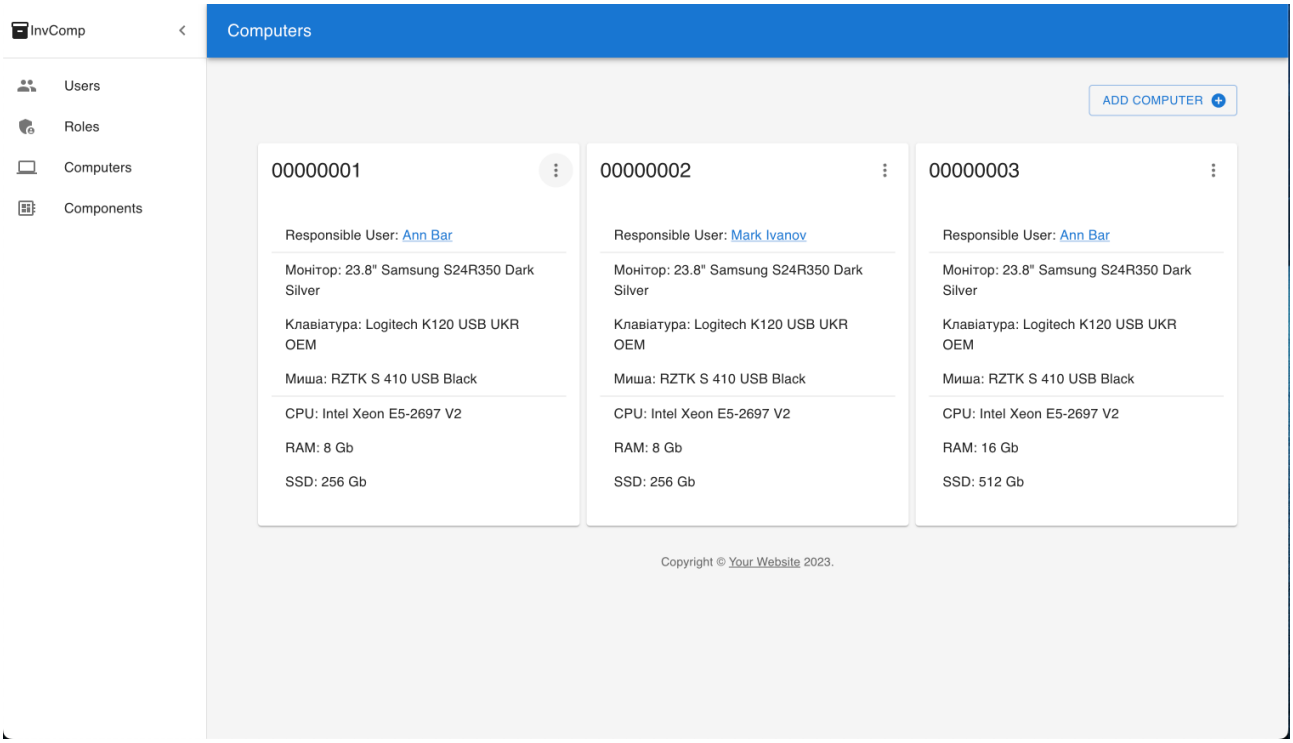

Рисунок 4.4 – Інформаційна панель на сторінці “Computers”

Натиснувши на кнопку “Computers” в меню зліва, користувач який має роль “Administrator” перейде до сторінки на якій зображено список комп’ютерів. На цій сторінці користувач має можливість додавати, редагувати та видаляти комп’ютери. Цифри зверху картки відображають інвентарний номер комп’ютеру. Натиснувши на ім’я користувача, користувач перейдо до детальної сторінки огляду користувача.

ВИСНОВКИ

У роботі була проведена комплексна розробка веб-додатку для обліку комп’ютерів і комплектуючих в організації. Під час роботи були розглянуті різні аспекти проєктування та реалізації веб-додатків, а також були використані сучасні технології та методології для забезпечення функціональності, надійності та швидкодії системи.

У результаті дослідження та аналізу було визначено, що використання клієнт-серверної архітектури, реляційної СКБД, RESTful API та MVC архітектури сприяє ефективному розробленню та підтримці веб-додатку. Ці підходи дозволяють забезпечити модульність, розширюваність та переносимість системи, а також спрощують розробку та тестування компонентів.

Детальний аналіз вимог до системи, проєктування архітектури та розробка інформаційної моделі забезпечують структурованість та організацію процесу розробки. Використання діаграм прецедентів та діаграм класів дозволяє чітко визначити функціональність системи, її взаємодію та структуру компонентів.

Процес розробки веб-додатку включав етапи від аналізу вимог до реалізації та тестування системи. Використання сучасних технологій, таких як JavaScript фреймворки (наприклад, React.js), реляційні СКБД (такі як MySQL та PostgreSQL) та використання практик Agile розробки дозволили забезпечити високу якість та швидкість розробки.

В результаті роботи над проєктом був розроблений функціональний веб-додаток, який дозволяє ефективно вести облік комп’ютерів і комплектуючих в організації. Додаток має інтуїтивний інтерфейс, а також надає зручні засоби пошуку та фільтрації даних. Реалізація додатку відповідає сучасним вимогам до надійності, безпеки та продуктивності веб-додатків.

Отже, дана дипломна робота є вагомим внеском у сферу розробки веб-додатків та надає практичні рекомендації та методології для ефективної реалізації подібних проєктів. Результати дослідження можуть бути використані

як основа для подальшого розвитку та вдосконалення веб-додатків у сфері управління інвентарем комп'ютерної техніки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Peretti A., Patrick H. Nest.js: A Progressive Node.js Framework. Santa Rosa : Bleeding Edge Press, 2018. 317 p.

2. Ford N., Richards M., Sadalage P. Software Architecture: The Hard Parts. 1st Ed. Sebastopol : O'Reilly Media, 2021. 450 p.

3. Obe R., Hsu L. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database 3rd Edition. Sebastopol : O'Reilly Media, 2017. 312 p.

4. IRMA Information Resources Management Association Web Application Development: Concepts, Methodologies, Tools, and Techniques. Hershey : IGI Global, 2015. 1788 p.

5. Hayward J., Fedosejev A., Prusty N., Horton A., Vice R., Holmes E., Bray T. React: Building Modern Web Applications. Birmingham : Packt Publishing, 2016. 890 p.

6. Manzoor A. Relational Databases: Design, Implementation, and Application Development. Scotts Valley : CreateSpace Independent Publishing Platform, 2012. 298 p.

7. Molinaro A. SQL Cookbook: Query Solutions and Techniques for Database Developers. Sebastopol : O'Reilly Media, 2005. 636 p.

8. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley Professional, 2002. 560 p.

9. Newman S. Building Microservices: Designing Fine-Grained Systems. Sebastopol : O'Reilly Media, 2015. 280 p.

10. Gamma E., Helm R., Johnson R., Vlissides J., Booch G. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley Professional, 1994. 416 p.