ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота

на тему: «**ВЕБ-ДОДАТОК ПІЗНАВАЛЬНОЇ ГРИ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ»**

на здобуття освітнього ступеня магістра
зі спеціальності __122 Комп'ютерні науки_____
(код, найменування спеціальності)
освітньо-професійної програми __Комп'ютерні науки _____
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ ___Ілля ПРОЦЕНКО_____
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. КНДМ - 63

_____Ілля ПРОЦЕНКО_____
(Ім'я, ПРІЗВИЩЕ)

Керівник: д.ф., Петро КРАВЧУК_____
Науковий ступінь, вчене звання *(Ім'я, ПРІЗВИЩЕ)*

Рецензент: _____
Науковий ступінь, вчене звання *(Ім'я, ПРІЗВИЩЕ)*

Київ – 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально–науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук
Ступінь вищої освіти − «Магістр»
Спеціальність підготовки − «122 Комп'ютерні науки»
Освітньо-професійна програма − «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____Віктор Вишнівський
"____" _____2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Проценко Ілля Юрійович
(прізвище, ім'я, по батькові)

1.Тема кваліфікаційної роботи: «Веб-додаток пізнавальної гри на основі штучного інтелекту»
Керівник кваліфікаційної роботи __ Петро Кравчук, д.ф.,
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу № 320 від 15.10.2024р.
2. Строк подання студентом роботи 15.12.2024 р.
3. Вихідні дані до роботи: Технології веб-розробки, Штучний інтелект, Функціонал гри, Адаптивність додатку. Науково-технічна література з питань, пов'язаних з побудовою мікросервісної архітектури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Сучасні підходи до розробки веб-додатків із використанням штучного інтелекту.
2. Аналіз платформ і фреймворків для створення інтерактивних пізнавальних ігор.
3. Проєктування структури веб-додатку: інтеграція фронтенду, бекенду та модуля штучного інтелекту.
4. Використання алгоритмів машинного навчання для адаптації рівнів складності гри.
5. Реалізація динамічної генерації запитань на основі вибраної тематики.

6. Оптимізація взаємодії користувача з додатком: використання гейміфікації та UX-дизайну.

7. Забезпечення надійності та безпеки даних користувачів у веб-додатку.

8. Тестування та оптимізація продуктивності: аналіз часу завантаження та відгуку додатку.

9. Висновки за результатами розробки системи.

5. Перелік ілюстративного матеріалу: *презентація*

1. Мета роботи, об`єкт та предмет дослідження
2. Архітектурна модель веб-додатку для пізнавальної гри на основі штучного інтелекту
3. Інтерактивна структура інтерфейсу користувача: екрани гри, меню та підказки
4. Схема взаємодії модулів штучного інтелекту з бекендом та фронтендом
5. Використання алгоритмів машинного навчання для адаптації рівнів складності гри
6. Процес генерації запитань та аналізу відповідей на основі NLP (Natural Language Processing)
7. Механізми забезпечення безпеки даних користувачів у веб-додатку
8. Візуалізація результатів тестування додатку (продуктивність, час відповіді сервера)
9. Висновки та подальші перспективи розвитку проєкту

6. Дата видачі завдання 05.09.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	08.10.2024	вик.
2	Аналіз сучасних підходів до інтеграції штучного інтелекту у пізнавальні ігри	22.10.2024	вик.
3	Проєктування архітектури веб-додатку та його основних модулів	05.11.2024	вик.
4	Розробка алгоритмів адаптації складності гри та аналізу відповідей	24.11.2024	вик.
5	Реалізація функціональних компонентів веб-додатку	09.12.2024	вик.
6	Вступ, висновки, реферат	11.12.2024	вик.
7	Розробка демонстраційних креслень	14.12.2024	вик.

Здобувач вищої освіти _____ Ілля ПРОЦЕНКО
(підпис) (Ім'я, ПРІЗВИЩЕ)

Керівник
кваліфікаційної роботи _____ Петро КРАВЧУК
(підпис) (Ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 80 стор., 1 рис., 20 джерел.

Мета роботи – розробка концептуальної моделі веб-додатку пізнавальної гри, яка базується на інтеграції штучного інтелекту для забезпечення інтерактивності, зручності та масштабованості.

Об`єкт дослідження – процеси розробки веб-додатків з інтерактивними та пізнавальними функціями на основі ШІ.

Предмет дослідження – методи проєктування та оптимізації архітектури веб-додатку з використанням сучасних технологій.

Короткий зміст роботи:

Магістерська робота присвячена розробці теоретичної та практичної моделі веб-додатку пізнавальної гри на основі штучного інтелекту. У роботі проаналізовано сучасні підходи до інтеграції штучного інтелекту у веб-додатки, особливості використання алгоритмів машинного навчання для динамічної адаптації складності гри та генерації завдань. Розглянуто переваги та недоліки існуючих рішень у сфері освітніх ігор, визначено ключові вимоги до створення ефективного інтерфейсу та взаємодії з користувачами.

Результатом роботи є розроблена теоретична модель веб-додатку, а також рекомендації щодо практичного впровадження пізнавальної гри з інтегрованими модулями штучного інтелекту. Додаток відповідає сучасним вимогам до адаптивності, інтерактивності та надійності, що сприяє підвищенню ефективності освітніх процесів.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, ПІЗНАВАЛЬНА ГРА, ШТУЧНИЙ ІНТЕЛЕКТ, МАШИННЕ НАВЧАННЯ, NATURAL LANGUAGE PROCESSING (NLP), ГЕЙМІФІКАЦІЯ, АДАПТИВНІСТЬ.

ABSTRACT

Text part of the master's qualification work: 80 pages, 1 pictures, 20 sources.

The purpose of the work – development of a conceptual model for an educational game web application based on the integration of artificial intelligence to ensure interactivity, usability, and scalability.

Object of research – the processes of developing web applications with interactive and educational functions based on AI.

Subject of research – methods of designing and optimizing the architecture of a web application using modern technologies.

Summary of the work:

The master's thesis is dedicated to the development of a theoretical and practical model of an educational game web application based on artificial intelligence. The study analyzes modern approaches to integrating artificial intelligence into web applications, the peculiarities of using machine learning algorithms for dynamic adaptation of game complexity and task generation. The advantages and disadvantages of existing solutions in the field of educational games are reviewed, and the key requirements for creating an effective interface and user interaction are determined.

The result of the work is a theoretical model of a web application, as well as recommendations for the practical implementation of an educational game with integrated artificial intelligence modules. The application meets modern requirements for adaptability, interactivity, and reliability, contributing to the enhancement of educational processes.

KEYWORDS: WEB APPLICATION, EDUCATIONAL GAME, ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, NATURAL LANGUAGE PROCESSING (NLP), GAMIFICATION, ADAPTABILITY.

ЗМІСТ

ВСТУП

Розвиток сучасних технологій, зокрема штучного інтелекту (ШІ), відкриває нові можливості для створення інноваційних веб-додатків. Такі додатки здатні не лише забезпечувати високий рівень інтерактивності, а й пропонувати користувачам унікальний досвід взаємодії. Особливе місце серед цих розробок займають пізнавальні ігри, які поєднують розважальні та освітні елементи. Вони сприяють розвитку критичного мислення, навичок аналізу інформації, творчого підходу до вирішення задач і дозволяють користувачам вивчати нові знання через інтерактивну взаємодію. Розробка таких додатків стає важливим інструментом у сфері освіти, дозвілля та підвищення кваліфікації в різних галузях. Крім того, такі додатки можуть слугувати платформою для вивчення нових технологій, інструментів програмування та стимулювати інноваційний розвиток.

Веб-додатки, створені на основі сучасних фреймворків і технологій, таких як Angular і NestJS, забезпечують зручність розробки, масштабованість та стабільність у роботі. Angular дозволяє створювати динамічний інтерфейс користувача із високою швидкодією, а NestJS надає структуровану основу для серверної логіки, забезпечуючи модульність та легкість у підтримці коду. Ці технології сприяють прискоренню розробки, полегшенню оновлення додатку та зменшенню витрат на підтримку. Інтеграція бібліотек для роботи з WebSocket, таких як socket.io, дозволяє створювати інтерактивну взаємодію між користувачами та сервером у режимі реального часу. Це особливо актуально для додатків, що потребують швидкої реакції системи, таких як ігри, освітні симулятори чи бізнес-додатки, що підтримують співпрацю в реальному часі. У даному контексті використання баз даних Elasticsearch значно підвищує ефективність роботи з великими обсягами даних завдяки швидкому індексуванню та пошуку. Elasticsearch дозволяє забезпечувати миттєвий доступ до необхідної інформації навіть у масштабних проєктах, таких як платформи для багатокористувацьких ігор або освітніх платформ. Крім того, підключення Claude API як засобу штучного інтелекту додає додатку можливість аналізувати дії користувачів, прогнозувати їхні потреби та адаптувати ігровий процес під індивідуальні вимоги. Це відкриває нові горизонти для персоналізації користувацького досвіду, створення унікальних сценаріїв взаємодії та розвитку гейміфікації.

Використання баз даних Elasticsearch значно підвищує ефективність роботи з великими обсягами даних завдяки швидкому індексуванню та пошуку. Elasticsearch дозволяє забезпечувати миттєвий доступ до необхідної інформації навіть у масштабних проєктах. Крім того, підключення Claude API як засобу штучного інтелекту додає додатку можливість аналізувати дії користувачів, прогнозувати їхні потреби та адаптувати ігровий процес під індивідуальні вимоги. Це відкриває нові горизонти для персоналізації користувацького досвіду та створення унікальних сценаріїв взаємодії.

Актуальність даного дослідження обумовлена зростаючою потребою у високоякісних пізнавальних веб-додатках, які поєднують інтерактивність, гнучкість та функціональність. Такі додатки мають широкий спектр застосування: від освітніх платформ до розважальних сервісів.

Метою роботи є розробка концептуальної моделі веб-додатку пізнавальної гри, яка базується на інтеграції штучного інтелекту для забезпечення інтерактивності, зручності та масштабованості.

Об'єктом дослідження є процеси розробки веб-додатків з інтерактивними та пізнавальними функціями на основі ШІ.

Предметом дослідження є методи проєктування та оптимізації архітектури веб-додатку з використанням сучасних технологій.

Запропоновано інтеграцію технологій штучного інтелекту для підвищення адаптивності пізнавальних веб-додатків, а також обґрунтовано використання Elasticsearch як ефективного засобу зберігання та пошуку даних у таких системах.

Введення нових технологій у процес розробки пізнавальних веб-додатків дозволяє створювати інноваційні продукти, які поєднують високу інтерактивність, адаптивність до потреб користувачів та можливість подальшого вдосконалення. Такі рішення здатні значно розширити можливості у сферах освіти, розваг і бізнесу, сприяючи не лише ефективності навчання, але й розвитку критичного мислення, підвищенню мотивації та залученості користувачів. Це дослідження спрямоване на розвиток підходів, які інтегрують сучасні технології та інструменти для створення продуктів, що відповідають високим вимогам ринку, забезпечуючи їхню конкурентоспроможність і перспективність..

1 АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ТА ВИБІР ТЕХНОЛОГІЙ

1.1 Сучасні тренди у розробці пізнавальних ігор

У сучасному світі пізнавальні ігри набувають усе більшої популярності, оскільки вони поєднують освітні, розважальні та тренувальні елементи. Їхня мета полягає у розвитку ключових компетенцій, таких як логічне мислення, пам'ять, креативність, здатність вирішувати складні задачі та аналітичні навички. Окрім того, ці ігри сприяють підвищенню обізнаності у різних сферах знань, зокрема науки, культури та технологій. Завдяки інтеграції сучасних технологій, пізнавальні ігри також забезпечують динамічну адаптацію до індивідуальних потреб кожного користувача, що стимулює тривалу мотивацію до навчання та саморозвитку. Важливим є те, що такі ігри залучають елементи гейміфікації, які не лише роблять навчання цікавим, але й забезпечують емоційне залучення, що сприяє більш глибокому засвоєнню інформації.

Технології, що використовуються для створення таких ігор, постійно еволюціонують, дозволяючи розробникам створювати більш складні, інтерактивні та адаптивні рішення. Зокрема, значна увага приділяється впровадженню алгоритмів штучного інтелекту, які можуть аналізувати величезні масиви даних, приймати рішення в режимі реального часу та забезпечувати адаптацію до індивідуальних потреб кожного гравця. Наприклад, такі алгоритми дають змогу створювати персоналізовані траєкторії навчання, підлаштовувати рівень складності гри, враховуючи успіхи чи труднощі гравця, та надавати відповідний зворотний зв'язок. Крім того, технології ШІ здатні аналізувати тривалі тенденції в поведінці користувачів, що дозволяє розробникам краще зрозуміти потреби аудиторії та вдосконалювати сценарії гри. Завдяки цьому ігри стають не лише джерелом розваг, але й потужним освітнім інструментом, що формує глибший рівень залучення користувачів. Інтеграція з елементами аналітики та навчання сприяє створенню динамічного ігрового середовища, яке постійно змінюється та адаптується до потреб гравців, роблячи кожен досвід унікальним. Впровадження

багатокористувацьких функцій стає ключовим аспектом, що забезпечує соціалізацію ігрового процесу. Сучасні платформи дозволяють організувати кооперативні ігри, змагання або інтерактивне навчання у великих групах, що значно підвищує залученість користувачів. Завдяки цьому гравці можуть не лише змагатися між собою, а й спільно вирішувати складні завдання, обмінюватися знаннями та ідеями, створюючи унікальну спільноту. Крім того, інтеграція сучасних платформ, таких як хмарні обчислення, дозволяє обробляти великі обсяги даних у реальному часі, забезпечуючи безперебійну роботу навіть у пікові моменти активності. Це відкриває можливість масштабування проєктів до глобального рівня, залучаючи мільйони користувачів і підтримуючи їхню взаємодію незалежно від географічного розташування.

Одним із ключових трендів є використання штучного інтелекту (ШІ). ШІ дозволяє аналізувати поведінку користувача, адаптувати складність гри під його рівень, а також пропонувати персоналізовані рекомендації. Завдяки алгоритмам машинного навчання система може визначати оптимальні підказки для гравців, виявляти закономірності у їхній поведінці, а також прогнозувати можливі дії користувачів. Крім того, ШІ здатний створювати унікальні сценарії гри, які відповідають індивідуальним особливостям кожного користувача, підвищуючи його залученість та інтерес. Впровадження технологій ШІ у пізнавальні ігри також дозволяє автоматично генерувати новий контент, адаптувати графіку, аудіо та навіть сюжет під потреби гравця, що робить ігровий процес ще більш персоналізованим та захоплюючим.

Ще одним важливим напрямком є інтерактивність. За допомогою технологій WebSocket (наприклад, socket.io) розробники створюють ігри з підтримкою багатокористувацької взаємодії в реальному часі. Це сприяє покращенню досвіду користувачів, адже вони можуть спілкуватися та співпрацювати один з одним без затримок. Інтерактивність також відкриває можливості для створення унікальних сценаріїв гри, де дії одного гравця впливають на результати інших, що додає елемент соціальної взаємодії та конкуренції. Завдяки socket.io реалізується не лише

текстова чи голосова комунікація, але й спільна робота над завданнями, обмін ресурсами чи ігровими об'єктами.

WebSocket дозволяє миттєво передавати дані про ігровий прогрес, що критично для змагань або складних кооперативних ігор. Це дає змогу гравцям миттєво реагувати на зміни у грі, а розробникам забезпечувати безперервність ігрового процесу навіть у сценаріях із високим навантаженням. Завдяки WebSocket можна створювати складні механізми синхронізації, які підтримують інтерактивність навіть при великій кількості одночасних користувачів. Поєднання цих технологій із сучасними UI-елементами, такими як плавні анімації, динамічні інтерфейси та інтегровані графічні ефекти, створює глибокий занурювальний досвід. Такі елементи дозволяють користувачам відчути себе частиною ігрового світу, сприяючи емоційному залученню та тривалому утриманню уваги. У результаті це стимулює розвиток як ігрових, так і освітніх продуктів, що відповідають сучасним потребам ринку.

Значну роль відіграє використання хмарних технологій і масштабованих баз даних, таких як Elasticsearch. Завдяки цій технології можна обробляти великі обсяги даних у реальному часі, забезпечуючи ефективне управління інформацією. Elasticsearch дозволяє зберігати історії ігрових процесів, статистику гравців, логи дій та інші важливі метрики. Такий підхід сприяє створенню персоналізованого досвіду для кожного користувача, оскільки система швидко надає доступ до необхідної інформації, базуючись на попередніх взаємодіях. Крім того, Elasticsearch підтримує складні пошукові запити та аналітику, що дозволяє розробникам отримувати цінну інформацію для вдосконалення ігрових механік. Використання цієї бази даних також забезпечує масштабованість, дозволяючи іграм працювати стабільно навіть за умов великої кількості активних користувачів.

На сучасному етапі розробки пізнавальних ігор важливим є створення адаптивних та модульних рішень, які легко вдосконалювати та підтримувати. Використання таких фреймворків, як Angular для клієнтської частини та NestJS для серверної, сприяє організації чіткої структури додатків і полегшує інтеграцію

нових функцій. Angular дозволяє створювати динамічні та інтерактивні інтерфейси користувача, які забезпечують високу продуктивність і зручність використання. NestJS, своєю чергою, забезпечує модульність і можливість швидкої розробки складних серверних систем. Інтеграція API, таких як Claude API, дозволяє реалізувати функціонал штучного інтелекту, що сприяє персоналізації ігрового процесу. Claude API може використовуватися для створення інтелектуальних чатботів, аналізу дій користувачів і формування рекомендацій у реальному часі. Наприклад, API може допомогти відслідковувати стиль гри користувача, адаптувати рівень складності завдань або автоматично підлаштовувати сценарії гри для підвищення її цікавості. Крім цього, Claude API може бути використаний для генерації динамічного контенту, такого як нові запитання чи рівні, що підвищує унікальність ігрового досвіду. Завдяки цим можливостям гра не лише стає більш інклюзивною, але й дозволяє враховувати різні потреби й уподобання гравців, що є критично важливим у сучасних розробках.

Такі підходи дозволяють створювати проєкти, які є масштабованими, простими у підтримці й здатними швидко адаптуватися до змін потреб ринку чи вимог користувачів. Завдяки цьому, пізнавальні ігри можуть інтегруватися у широкі екосистеми, об'єднуючи освітні, розважальні та тренувальні функції. Крім того, такі проєкти можуть слугувати платформою для експериментів із новими технологіями, зокрема віртуальною та доповненою реальністю, які ще більше підвищують рівень занурення користувачів.

У результаті, пізнавальні ігри стають не лише ефективним освітнім інструментом, а й потужним засобом розваг, що забезпечує високу якість користувацького досвіду. Вони сприяють популяризації сучасних технологій, таких як штучний інтелект, віртуальна та доповнена реальність, а також забезпечують інтеграцію з інноваційними платформами для навчання. Окрім цього, пізнавальні ігри сприяють формуванню важливих навичок XXI століття, серед яких критичне мислення, креативність, навички співпраці, лідерства та адаптивності. Застосування таких ігор дозволяє не лише отримувати нові знання, але й активно

використовувати їх у життєвих та професійних ситуаціях, формуючи основу для постійного розвитку.

Сучасні тренди у розробці пізнавальних ігор орієнтовані на інтерактивність, персоналізацію, масштабованість та адаптивність. Вони не лише вдосконалюють освітні процеси, але й сприяють залученню нових аудиторій завдяки інноваційним технологіям, що робить їх потужним інструментом для популяризації сучасних знань і навичок у розважальному форматі. Розробка таких ігор відповідає запитам ринку, адже забезпечує високий рівень інтеграції технологій, взаємодії користувачів і можливість постійного розвитку на основі зворотного зв’язку..

1.2 Вибір технологій для проєкту

У процесі вибору технологій для розробки веб-додатку пізнавальної гри ключову роль відіграють вимоги до інтерактивності, масштабованості, продуктивності, гнучкості та можливості інтеграції з сучасними платформами. Окрім цього, технології мають забезпечувати підтримку інноваційних функцій, таких як автоматизація навчальних процесів, інтерактивні сценарії, можливість персоналізації користувацького досвіду та швидку адаптацію до змінних вимог ринку. Технології також повинні забезпечувати ефективну роботу в умовах високого навантаження та підтримувати комплексну взаємодію з іншими системами.

Серед важливих аспектів вибору технологій є їх гнучкість у реалізації складних алгоритмів та відповідність принципам модульного проєктування, що дозволяє адаптувати продукт до потреб зростаючої аудиторії. Інтеграція із системами штучного інтелекту відкриває можливість для реалізації таких функцій, як динамічна зміна контенту, аналіз поведінки користувачів та автоматичне налаштування складності гри. Крім того, використання сучасних рішень для роботи з великими обсягами даних дозволяє ефективно організовувати їх обробку та зберігання, що є критично важливим для масштабованих веб-додатків. Зважаючи на ці вимоги, обрано наступні технології:

Angular - цей фронтенд-фреймворк забезпечує створення динамічного, інтерактивного та адаптивного інтерфейсу користувача. Його ключові переваги включають:

- Швидкодія завдяки двосторонньому зв’язку даних (two-way data binding), що забезпечує миттєве оновлення інтерфейсу при зміні даних.
- Модульна структура, яка спрощує розробку та дозволяє легко додавати нові компоненти без значних змін у вже існуючому коді. Це забезпечує зручність масштабування та підтримки проєкту.
- Розширені можливості для роботи з анімаціями, інтерактивними формами та динамічним відображенням даних. Angular дозволяє створювати багатий візуальний досвід, який підвищує залучення користувачів.
- Вбудовані інструменти для тестування, які забезпечують високу якість коду та стабільність додатків. Завдяки цим інструментам розробники можуть легко виявляти та виправляти помилки на ранніх етапах розробки.
- Потужна екосистема бібліотек та розширень, що дає змогу адаптувати Angular до будь-яких потреб проєкту. Angular підтримує інтеграцію з популярними інструментами розробки, такими як RxJS для роботи з асинхронними даними, і має велику спільноту, яка постійно вдосконалює інструменти та бібліотеки.

NestJS - серверний фреймворк на базі Node.js, який пропонує:

- Модульний підхід до побудови архітектури, що спрощує розширення, оновлення та підтримку системи навіть для великих проєктів.
- Високу продуктивність та масштабованість завдяки використанню JavaScript та TypeScript, що дозволяє забезпечити стабільну роботу навіть за умов великої кількості одночасних запитів. Це дозволяє створювати веб-додатки, здатні обробляти мільйони запитів на день.

- Інтеграцію з іншими сервісами через REST API або WebSocket, а також підтримку GraphQL для гнучкішої роботи з даними. GraphQL забезпечує можливість точного вибору даних, що значно підвищує продуктивність клієнт-серверної взаємодії.
- Вбудовану підтримку декораторів та інверсії контролю (Inversion of Control), що сприяє чистій, організованій та зрозумілій структурі коду. Це спрощує впровадження принципів Dependency Injection, що є основою для гнучкого і масштабованого дизайну додатків.
- Потужну екосистему модулів, яка включає рішення для аутентифікації, валідації, роботи з базами даних, управління чергами завдань (task queues), кешування та обробки асинхронних запитів. Завдяки цьому NestJS стає ідеальним вибором для високонавантажених додатків.

Socket.io - бібліотека для роботи з WebSocket, що дозволяє реалізувати функціонал взаємодії в реальному часі, зокрема:

- Чат між гравцями, включаючи текстові та голосові повідомлення для підвищення інтерактивності гри. Socket.io також підтримує реакції, файлообмін та створення групових чатів.
- Оновлення даних на екрані без перезавантаження сторінки, що забезпечує плавність і безперервність користувацького досвіду. Це особливо важливо для динамічних ігор, де кожна секунда має значення.
- Синхронізацію ігрового процесу в багатокористувацькому режимі, дозволяючи гравцям миттєво взаємодіяти з іншими учасниками. Система також підтримує управління сесіями та збереження ігрового прогресу в режимі реального часу.
- Підтримку широкомасштабних ігрових сесій із сотнями або тисячами користувачів одночасно, забезпечуючи стабільну продуктивність навіть за умов пікових навантажень.

- Гнучкі механізми для обробки подій, які дозволяють легко керувати складними сценаріями взаємодії між гравцями, такими як створення альянсів, змагань, спільних квестів або унікальних командних завдань.

Claude API - потужний інструмент для інтеграції штучного інтелекту, який надає:

- Можливості для генерації рекомендацій та адаптації ігрового процесу, що дозволяє створювати унікальний ігровий досвід для кожного користувача. Система здатна адаптувати сценарії гри на основі аналізу історії взаємодії користувача, підвищуючи його мотивацію.
- Аналіз дій користувачів для покращення ігрового досвіду шляхом виявлення сильних і слабких сторін гравця, адаптації рівнів складності та створення більш інтуїтивного інтерфейсу. Claude API може виявляти поведінкові патерни та прогнозувати можливі дії користувачів, пропонуючи оптимальні підказки чи рекомендації.
- Інструменти для автоматизації навчальних елементів у грі, таких як інтерактивні підказки, адаптивні завдання та генерація індивідуальних навчальних траєкторій. Ці функції роблять гру ефективним освітнім інструментом.
- Підтримку діалогових систем для створення інтелектуальних ботів, які можуть взаємодіяти з гравцями в реальному часі, підвищуючи їхню залученість. Боти здатні адаптуватися до стилю спілкування користувача, створюючи більш природну та інтуїтивну взаємодію.
- Інтеграцію інструментів для обробки природної мови, що дозволяє проводити аналіз текстових повідомлень, оцінювати емоційний стан гравців та відповідно налаштовувати ігровий процес.

Elasticsearch - база даних, яка забезпечує:

- Швидкий пошук великих обсягів даних, дозволяючи миттєво обробляти запити навіть при великих навантаженнях. Elasticsearch

використовує потужну індексацію, що гарантує мінімальні затримки під час обробки даних.

- Індексацію ігрових профілів, результатів, історії гравців, а також іншої метаданих, що сприяє персоналізації ігрового досвіду. Завдяки цьому можлива сегментація гравців за різними параметрами для кращого розуміння їхніх потреб.
- Масштабованість для обробки зростаючої кількості користувачів, забезпечуючи стабільну продуктивність навіть за умов пікової активності. База даних легко адаптується до зростаючих потреб проєкту завдяки кластеризації.
- Інтеграцію з аналітичними інструментами для моніторингу даних у реальному часі, що дозволяє оперативно реагувати на потреби гравців та оптимізувати ігровий процес. Це включає автоматичне сповіщення про аномалії у даних або поведінці користувачів.
- Гнучкість у налаштуванні пошукових запитів і фільтрів, що дозволяє створювати інтуїтивні системи навігації та доступу до контенту. Elasticsearch підтримує запити природною мовою, що значно спрощує взаємодію користувачів із системою.
- Розширені можливості аналітики, які включають побудову дашбордів для візуалізації даних, аналіз поведінкових патернів гравців та прогнозування результатів.

Дані технології були обрані через їх відповідність вимогам проєкту, здатність забезпечувати надійну роботу у високонавантажених умовах, адаптивність до змінних потреб користувачів і ефективність у реалізації складних інтерактивних сценаріїв. Вони дозволяють створювати продукти, які легко масштабуються, підтримують інтеграцію з різноманітними API та сервісами, і забезпечують швидке впровадження нових функцій. Їхнє поширене використання в сучасних розробках також гарантує доступність великої кількості ресурсів для навчання, активну підтримку спільнот, широкий вибір готових рішень та бібліотек, що значно

спрощує процес впровадження. Крім того, ці технології пропонують інструменти для вдосконалення продуктивності, аналізу даних у реальному часі та забезпечення високого рівня безпеки, що критично важливо для сучасних інтерактивних додатків.

1.3 Використання Elasticsearch для роботи з даними: переваги і недоліки

Elasticsearch є однією з найпоширеніших сучасних баз даних для роботи з великими обсягами інформації. Її популярність пояснюється здатністю забезпечувати надзвичайно швидкий доступ до даних завдяки високоефективній системі індексації, яка забезпечує обробку мільйонів записів за лічені секунди. Це критично важливо для інтерактивних додатків, пізнавальних ігор, аналітичних платформ і систем рекомендацій, де швидкість доступу до інформації є ключовим фактором. Elasticsearch забезпечує не тільки оперативність, але й надійність у виконанні пошукових запитів, гарантуючи високу точність результатів навіть у системах із величезними наборами даних. Її продуктивність дозволяє створювати додатки, які працюють плавно та ефективно, забезпечуючи користувачів найкращим досвідом.

Однією з найважливіших характеристик Elasticsearch є її масштабованість. Ця особливість дозволяє динамічно збільшувати обчислювальні потужності системи шляхом додавання нових вузлів до кластеру. Завдяки такому підходу система може адаптуватися до зростаючих навантажень, забезпечуючи стабільну продуктивність навіть під час пікових періодів. Наприклад, у масштабних проєктах, таких як багатокористувацькі онлайн-ігри чи системи обробки даних електронної комерції, масштабованість дозволяє підтримувати синхронізацію дій тисяч користувачів і забезпечувати швидку обробку пошукових запитів. Elasticsearch також підтримує автоматичне балансування навантаження між вузлами, що додатково сприяє стабільності роботи системи, навіть за умов значного збільшення обсягу оброблюваних даних. Завдяки цьому ця технологія

стає критично важливою для підтримки конкурентоспроможності сучасних інтерактивних додатків.

Завдяки своїй архітектурі Elasticsearch забезпечує високу продуктивність навіть у найскладніших сценаріях використання. Наприклад, у багатокористувацьких іграх платформа дозволяє синхронізувати дії тисяч гравців у реальному часі, мінімізуючи затримки та забезпечуючи стабільний ігровий процес. У платформах електронної комерції Elasticsearch демонструє здатність обробляти мільйони пошукових запитів щодня, що є критичним під час великих розпродажів чи маркетингових акцій, коли обсяги трафіку зростають у кілька разів. Завдяки цьому Elasticsearch забезпечує безперебійну роботу систем, адаптуючись до змінних умов і навантажень. Така стабільність і масштабованість стають ключовими перевагами, які роблять Elasticsearch оптимальним вибором для великих, критично важливих проєктів, орієнтованих на високу продуктивність і надійність. Ще однією вагомою перевагою є підтримка індексації та обробки даних у реальному часі, що особливо важливо для динамічних середовищ. Elasticsearch забезпечує безперервне оновлення індексів, дозволяючи зберігати актуальність даних навіть за умов інтенсивного потоку інформації. Завдяки цій особливості технологія може обслуговувати сценарії з постійно змінюваними даними, такими як онлайн-ігри, аналітичні панелі чи системи рекомендацій. У контексті ігрових систем Elasticsearch дозволяє миттєво оновлювати результати, рейтинги або статуси гравців, підтримуючи інтерактивність та залученість користувачів. Крім того, її здатність інтегруватися з іншими інструментами, такими як Logstash або Beats, дозволяє автоматизувати збір і передачу даних, що значно підвищує ефективність роботи системи.

Значною перевагою Elasticsearch є гнучкість у побудові запитів, яка дозволяє створювати надзвичайно точні та адаптивні системи пошуку, що відповідають потребам навіть найвимогливіших користувачів. Завдяки підтримці складних пошукових запитів, які можуть включати фільтри, агрегати, сортування і навіть запити природною мовою, Elasticsearch відкриває широкі можливості для

глибокого аналізу даних. Такий підхід дозволяє забезпечити користувачам швидкий доступ до релевантної інформації, навіть у разі роботи з масивними наборами даних. Крім того, підтримка запитів природною мовою спрощує взаємодію з системою для користувачів без технічних знань, що робить Elasticsearch зручним вибором для створення інтуїтивно зрозумілих інтерфейсів і підвищує загальну задоволеність клієнтів. У поєднанні з адаптивними механізмами система здатна відповідати на різноманітні запити користувачів, оптимізуючи процес пошуку та підвищуючи ефективність роботи.

Додатково інтеграція з Kibana (рис. 1.1) надає потужні інструменти для аналітики, які дозволяють створювати детальні дашборди, що відображають ключові тренди, статистичні дані та інші важливі метрики. Завдяки цьому розробники та адміністратори системи можуть відслідковувати продуктивність у реальному часі, оцінювати вплив змін у системі та приймати зважені стратегічні рішення. Крім того, Kibana дає можливість проводити глибокий аналіз поведінкових патернів користувачів, виявляти тенденції та оптимізувати ігровий процес на основі цих даних. У пізнавальних іграх, наприклад, Kibana може показувати детальний прогрес гравців, аналізувати, які завдання вони виконують найкраще, і виділяти області для покращення. Це підвищує рівень персоналізації та сприяє більш інтерактивній взаємодії між системою та користувачем, що в кінцевому підсумку збільшує залученість і ефективність гри.

Проте Elasticsearch має і певні обмеження, які варто враховувати при його впровадженні. По-перше, система має високі вимоги до ресурсів: для забезпечення стабільної роботи потрібні значні обчислювальні потужності та велика кількість оперативної пам’яті, що може підвищити витрати на інфраструктуру. По-друге, процес налаштування Elasticsearch доволі складний, особливо для новачків, оскільки оптимізація параметрів роботи потребує досвіду і глибоких технічних знань. Окрім того, Elasticsearch не підтримує ACID-транзакції, що обмежує його застосування у сценаріях, де необхідно гарантувати абсолютну цілісність і консистентність даних, наприклад у фінансових додатках.

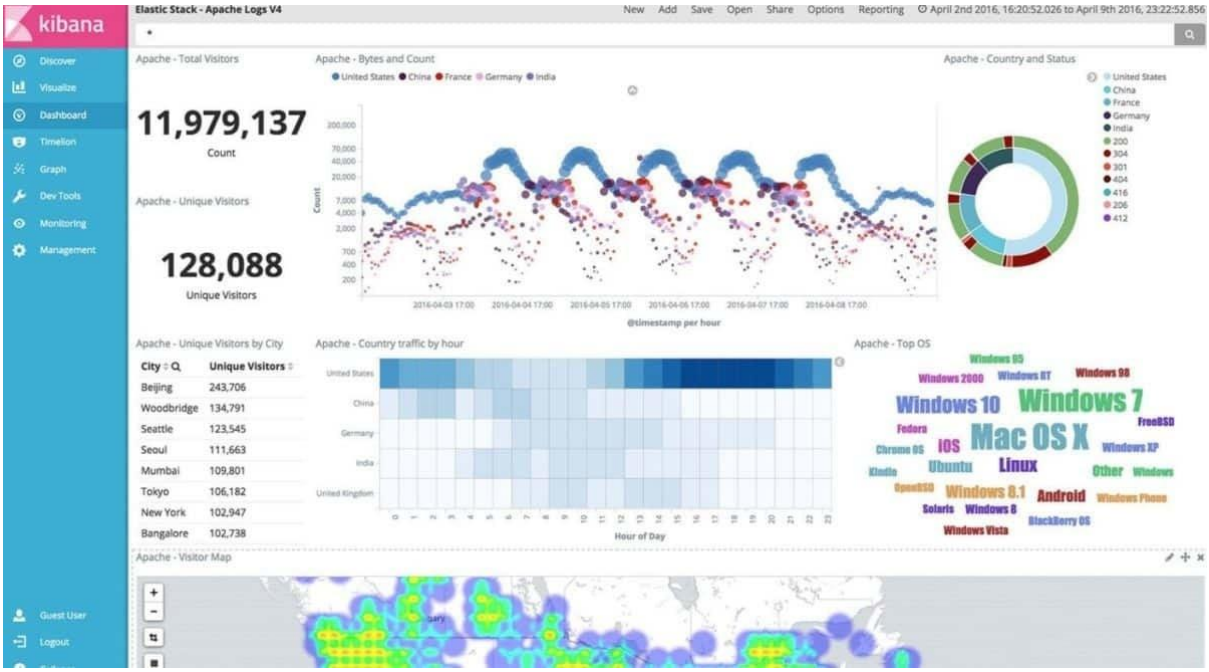

Рисунок 1.1 – Kibana

У пікові моменти навантаження, якщо кластери не налаштовані належним чином, продуктивність системи може значно знижуватися, що впливає на швидкість доступу до даних. Також слід зазначити, що відновлення після збоїв може бути ускладненим без регулярного резервного копіювання та належного моніторингу стану кластерів.

Elasticsearch є потужним та ефективним інструментом для роботи з великими обсягами даних, який забезпечує унікальну комбінацію швидкості, масштабованості та аналітичних можливостей. Його застосування є ідеальним вибором для інтерактивних додатків, особливо тих, що потребують обробки даних у реальному часі, підтримки складних пошукових запитів і динамічної взаємодії з користувачами. Elasticsearch дозволяє реалізувати гнучкі рішення для різних сценаріїв, включаючи багатокористувацькі системи, аналітичні панелі та системи рекомендацій. Проте, під час впровадження Elasticsearch необхідно враховувати ресурси, технічну складність налаштувань та особливості конкретного проєкту. Важливим аспектом є також регулярний моніторинг і резервне копіювання для забезпечення безперебійної роботи. Успішна інтеграція цієї технології значною

мірою залежить від кваліфікації команди, здатності адаптувати її до змінних вимог бізнесу та готовності інфраструктури до високих навантажень. Це робить Elasticsearch незамінним вибором для проєктів, які вимагають високої швидкості, надійності та інноваційного підходу до обробки даних..

1.4 Висновки по розділу

У першому розділі було проведено аналіз сучасних підходів до розробки пізнавальних ігор, визначено актуальні тренди у виборі технологій та обґрунтовано вибір інструментів для створення інтерактивних додатків. Зокрема, було акцентовано увагу на ключових характеристиках таких технологій, як Angular, NestJS, Socket.io, Claude API та Elasticsearch, які забезпечують високий рівень інтерактивності, продуктивності, масштабованості та адаптивності системи.

Розглянуто переваги та обмеження Elasticsearch як бази даних, яка є центральним елементом зберігання та обробки даних у проєкті. Визначено, що використання цієї технології сприяє реалізації складних пошукових запитів, забезпечує швидкість обробки даних і дозволяє масштабувати систему відповідно до потреб користувачів. Водночас наголошено на важливості врахування ресурсних вимог та складності налаштування при її інтеграції.

Виконаний аналіз дозволяє сформувати технологічну базу для подальшої реалізації проєкту, яка поєднує інноваційні технології, адаптивність до змінних потреб ринку та високу ефективність у вирішенні завдань користувачів. Це закладає основу для створення інтерактивного додатку, що забезпечує якісний досвід користувачів та підтримує вимоги сучасного програмного забезпечення..

2 ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ ПІЗНАВАЛЬНОЇ ГРИ

2.1 Концептуальна модель додатку та функціональні вимоги

Розробка концептуальної моделі веб-додатку пізнавальної гри є ключовим етапом у процесі проєктування, що формує основу всієї системи. Цей етап передбачає глибокий аналіз цільової аудиторії, зокрема її вікових характеристик, потреб та очікувань від продукту. У процесі створення концептуальної моделі визначаються ключові функції додатку, такі як доступність, зручність використання, інтерактивність та адаптивність. Побудова структурної схеми включає деталізацію всіх компонентів системи та їх інтеграції, що дозволяє уникнути проблем у подальшому розробленні. Важливим аспектом є формування чітких функціональних зв’язків між модулями, що забезпечує стабільність роботи додатку навіть за умов високих навантажень. Концептуальна модель забезпечує візуалізацію логіки роботи системи, полегшуючи розуміння її функціональних можливостей. Вона також спрощує комунікацію між командою розробників, дизайнерами, замовниками та майбутніми користувачами. Крім того, ця модель слугує базою для створення технічної документації, прототипування, планування тестування, оцінки ризиків та підготовки до масштабування проєкту.

Концептуальна модель включає наступні основні компоненти

Інтерфейс користувача (UI). Інтерактивний та адаптивний дизайн, що забезпечує зручність взаємодії з додатком. Ключові елементи інтерфейсу включають інформаційну панель, систему навігації, форми для введення даних, інтерактивні елементи гри, а також можливість персоналізації для кожного користувача. Реалізація персоналізації включає налаштування кольорової схеми, створення користувацьких профілів та інтеграцію мовних уподобань. Важливим є впровадження реактивних компонентів, які автоматично оновлюють дані без перезавантаження сторінки, що значно покращує динамічність ігрового процесу. Додатково інтерфейс забезпечує доступність для користувачів з обмеженими можливостями, включаючи масштабування шрифтів, голосові підказки та адаптацію для роботи на різних пристроях.

Серверна частина. Логіка додатку, яка відповідає за обробку даних, синхронізацію ігрового процесу між користувачами, захист від несанкціонованого доступу та обробку запитів у реальному часі. Вона включає обробку асинхронних запитів, оптимізацію кешування для зменшення затримок, а також забезпечення розподіленого управління ресурсами для стабільної роботи за високих навантажень. Серверна частина також взаємодіє з базою даних Elasticsearch для швидкого доступу до даних і підтримує інтеграцію API, таких як Claude API і Socket.io, що гарантує динамічну роботу системи з мінімальними затримками. Додатково, вона відповідає за моніторинг стану системи та виявлення потенційних збоїв для забезпечення надійності.

База даних. Elasticsearch, яка забезпечує зберігання, індексацію та швидкий доступ до інформації, такої як профілі користувачів, результати ігор, метадані та лог-файли. Її потужності дозволяють виконувати складні пошукові запити та аналіз даних у реальному часі, підтримуючи динамічну взаємодію між компонентами додатку. Elasticsearch оптимізує роботу системи завдяки своїй масштабованій архітектурі, забезпечуючи обробку великих обсягів даних без втрати продуктивності. Крім того, вона підтримує функції реплікації та автоматичного відновлення після збоїв, що підвищує надійність та стійкість додатку навіть у критичних умовах.

Інтеграція API. Claude API використовується для реалізації потужних алгоритмів штучного інтелекту, які адаптують ігровий процес до індивідуальних потреб кожного користувача, забезпечуючи персоналізацію і високий рівень залучення. Socket.io гарантує миттєву взаємодію між учасниками гри, дозволяючи реалізовувати багатокористувацький режим і синхронізувати ігрові події в реальному часі. Інтеграція API також включає використання додаткових інструментів для розширеної аналітики, моніторингу поведінкових патернів користувачів та оптимізації продуктивності системи. Це створює комплексний механізм підтримки гнучкої і стабільної роботи додатку, забезпечуючи зручність і ефективність як для користувачів, так і для розробників.

Функціональні вимоги

Реалізація веб-додатку базується на таких функціональних вимогах, які забезпечують його ефективність та відповідність сучасним стандартам програмного забезпечення:

Реєстрація та авторизація користувачів. Цей модуль дозволяє створювати нові облікові записи, входити в систему та керувати профілями користувачів. Застосування сучасних протоколів безпеки, таких як OAuth або JWT, гарантує захист особистих даних, шифрування паролів і запобігання несанкціонованому доступу. Крім того, модуль передбачає функції відновлення доступу, багатофакторної автентифікації та перевірки активності сеансів, що забезпечує додатковий рівень безпеки.

Ігровий процес. Основний акцент зроблено на інтерактивних завданнях, які адаптуються до рівня знань і досвіду кожного користувача. Завдяки використанню алгоритмів штучного інтелекту, система може автоматично коригувати складність завдань, пропонуючи оптимальний рівень виклику для кожного гравця. Кожне завдання доповнюється контекстуальною підтримкою, яка допомагає користувачам зрозуміти правила і цілі гри. Крім того, механізми гейміфікації, такі як нарахування балів, створення досягнень та інтерактивні підказки, сприяють підвищенню залученості та мотивації. Система дозволяє аналізувати прогрес користувачів у реальному часі та пропонувати рекомендації для покращення результатів.

Синхронізація даних у реальному часі. Всі ігрові події та дії користувачів оновлюються миттєво завдяки інтеграції з Socket.io. Це забезпечує безперебійну взаємодію між учасниками гри, створюючи динамічне середовище для змагань та співпраці. Socket.io дозволяє обмінюватися даними між клієнтами та сервером з мінімальною затримкою, що є ключовим для ігор, які потребують миттєвої реакції. Крім того, система підтримує відновлення сеансів у разі втрати підключення, що підвищує стабільність і комфорт користувачів. Інтеграція з іншими модулями,

такими як аналітика або керування сесіями, дозволяє розробникам створювати комплексний ігровий досвід.

Аналітика та статистика. Додаток надає розширені інструменти для візуалізації прогресу користувачів у вигляді графіків, таблиць та інтерактивних дашбордів. Зібрані дані дозволяють створювати детальні аналітичні звіти, які допомагають користувачам оцінювати свій розвиток, визначати сильні сторони та області для покращення. Розробники можуть використовувати ці дані для оптимізації ігрового процесу, вдосконалення механік гри та впровадження нових функцій. Система також підтримує аналіз поведінкових патернів користувачів, що дозволяє прогнозувати їхні дії та адаптувати контент під їхні потреби.

Масштабованість. Архітектура додатку розроблена таким чином, щоб забезпечити стабільну роботу навіть при різкому збільшенні кількості користувачів. Це досягається завдяки гнучкому масштабуванню серверів, інтеграції кластерів бази даних Elasticsearch та використанню автоматичного балансування навантаження. Elasticsearch дозволяє ефективно обробляти мільйони запитів у реальному часі, забезпечуючи швидкий доступ до даних і оптимізацію роботи. Завдяки цьому додаток залишається продуктивним і стійким навіть за умов пікових навантажень, що робить його надійним вибором для масштабних та довготривалих проєктів.

Дана концептуальна модель та визначені функціональні вимоги формують фундамент для створення веб-додатку, який відповідає найвищим стандартам сучасного програмного забезпечення. Вона не лише забезпечує гнучкість у розробці складних завдань, але й підтримує адаптивність системи до змінних потреб користувачів, швидкого розширення функціоналу та динамічного розвитку ринку. Завдяки глибокому аналізу і продуманому проєктуванню, система здатна інтегрувати передові технології, такі як штучний інтелект, машинне навчання і модулі для роботи в реальному часі, що значно підвищує загальний рівень користувацького досвіду. Модель створює умови для ефективного управління ресурсами, враховуючи баланс між високою продуктивністю та раціональним

використанням обчислювальних потужностей. Це забезпечує збереження стабільності системи навіть у періоди пікових навантажень. Можливість безперервного оновлення функціоналу та адаптації до змін на ринку робить цей веб-додаток не лише конкурентоспроможним, але й готовим до тривалого життєвого циклу.

Особливу увагу приділено підтримці масштабованості, що дозволяє розширювати обсяг оброблюваних даних та кількість одночасних користувачів без втрати продуктивності. Це досягається завдяки інтеграції з Elasticsearch, яка не лише підтримує складні запити, але й забезпечує високий рівень продуктивності навіть під час пікових навантажень. Завдяки архітектурі, що підтримує горизонтальне масштабування, Elasticsearch дозволяє легко збільшувати обчислювальні потужності шляхом додавання нових вузлів. Крім того, технологія забезпечує реплікацію даних для підвищення надійності та швидке відновлення після збоїв. У поєднанні з іншими компонентами моделі, такими як автоматичне балансування навантаження та оптимізація кешування, ця архітектура гарантує стабільну, швидкодійну і надійну роботу системи, що відповідає сучасним вимогам до програмного забезпечення.

Тому концептуальна модель є основою для створення не лише інноваційного продукту, а й стійкого, високопродуктивного рішення, яке повністю відповідає сучасним потребам і очікуванням користувачів. Її структура забезпечує не тільки гнучкість і адаптацію до технологічних змін, але й дозволяє інтегрувати нові функції, які покращують взаємодію користувачів із системою. Це рішення підтримує стабільність роботи, незалежно від навантаження, завдяки масштабованості та автоматизованим механізмам обробки даних. Модель гарантує довготривалий розвиток продукту, високу ефективність і здатність залишатися конкурентоспроможною на швидкозмінному ринку, забезпечуючи при цьому унікальний і персоналізований досвід для кожного користувача..

2.2 Архітектура системи: клієнт-серверна взаємодія

Архітектура системи пізнавальної гри побудована за принципом клієнт-серверної взаємодії, яка забезпечує чіткий розподіл функцій між клієнтською частиною та сервером. Цей підхід дозволяє створити гнучку, масштабовану та легко підтримувану систему, яка ефективно розподіляє ресурси, забезпечує високу продуктивність і знижує ризики перевантаження. Клієнтська частина реалізована з акцентом на швидкодію та інтерактивність, включаючи адаптивний інтерфейс, що змінюється в реальному часі відповідно до дій користувачів. Сервер, у свою чергу, відповідає за обробку великих обсягів даних, забезпечення стабільності при пікових навантаженнях та інтеграцію передових технологій, таких як штучний інтелект і машинне навчання. Такий розподіл дозволяє досягти високої ефективності системи, зберігаючи простоту обслуговування та можливість масштабування для підтримки зростаючої кількості користувачів.

Взаємодія між компонентами реалізується через стандартизовані протоколи, такі як HTTPS для захищеної передачі даних та WebSocket для миттєвої синхронізації. HTTPS забезпечує шифрування даних під час передачі, що мінімізує ризики перехоплення чи втрати інформації під час обміну між клієнтом і сервером. WebSocket дозволяє підтримувати постійний зв’язок між компонентами системи, забезпечуючи миттєві оновлення ігрових подій і синхронізацію користувацьких дій у режимі реального часу. Це сприяє надійності та безпеці передачі даних, захищаючи систему від можливих атак, таких як атаки типу «людина посередині» (MITM), та втрат інформації через збої у зв’язку. Така архітектура дозволяє легко адаптувати систему до змін вимог, інтегрувати нові технології, такі як машинне навчання для прогнозування ігрових сценаріїв, та масштабувати її для підтримки зростаючої кількості користувачів, гарантуючи стабільну роботу навіть за пікових навантажень.

Клієнтська частина відповідає за відображення інформації та взаємодію користувачів із системою. Вона реалізована за допомогою фреймворка Angular, який забезпечує динамічний і адаптивний інтерфейс користувача. Клієнтська

частина відповідає за генерацію UI-компонентів, адаптованих до дій користувача, і дозволяє інтегрувати реактивні бібліотеки для покращення швидкодії. Вона також підтримує інтернаціоналізацію, що робить додаток доступним для користувачів з різних країн. Основні функції клієнтської частини включають:

- Відправлення запитів до сервера через REST API або WebSocket для отримання чи надсилання даних, забезпечуючи швидку передачу інформації між клієнтом і сервером. REST API використовується для структурованих запитів і відповідає потребам асинхронного обміну даними, тоді як WebSocket забезпечує двосторонню комунікацію в режимі реального часу, необхідну для інтерактивного ігрового процесу та миттєвої синхронізації користувацьких дій.
- Обробка даних на рівні фронтенду включає попередню фільтрацію та агрегування інформації перед її передачею до сервера, що значно зменшує затримки. Це також забезпечує миттєвий відгук на дії користувача завдяки використанню реактивних бібліотек і оптимізованих алгоритмів рендерингу. Такий підхід не лише мінімізує навантаження на сервер, але й створює більш плавний і інтерактивний користувацький досвід, адаптуючи роботу додатку до умов реального часу.
- Реалізація інтерактивних компонентів, таких як таблиці, графіки, інтерактивні завдання з адаптивною візуалізацією, що змінюється в реальному часі. Компоненти підтримують динамічне оновлення даних, інтеграцію із зовнішніми сервісами для отримання актуальної інформації та взаємодію з користувачем через інтуїтивно зрозумілі інтерфейси. Такий підхід значно підвищує залученість користувачів і забезпечує гнучкість додатку.

Серверна частина, побудована на основі NestJS, виконує такі завдання:

- Обробка клієнтських запитів включає розподіл навантаження між запитами REST API, які забезпечують структурований доступ до даних,

та WebSocket, що гарантує безперервний зв’язок у реальному часі. Цей підхід дозволяє підтримувати інтерактивність додатку, швидко обробляти запити та забезпечувати стабільність навіть за умов високого навантаження. Додатково використовуються механізми кешування для зниження затримок і підвищення продуктивності передачі даних.

- Взаємодія з базою даних Elasticsearch забезпечує надійне зберігання великих обсягів даних, їх ефективну індексацію для швидкого пошуку та оперативне отримання необхідної інформації. Система використовує інтелектуальні алгоритми для автоматичної оптимізації запитів, що значно підвищує швидкодію та точність обробки запитів. Додатково Elasticsearch підтримує горизонтальне масштабування, що дозволяє зберігати стабільну продуктивність навіть за умов зростаючих навантажень.
- Реалізація бізнес-логіки включає адаптацію ігрових завдань на основі дій користувачів за допомогою Claude API, яке забезпечує динамічну генерацію завдань з урахуванням індивідуальних можливостей і прогресу кожного гравця. Claude API використовує штучний інтелект для аналізу поведінки користувачів, прогнозування їхніх дій та пропонування контекстуально релевантних завдань. Це сприяє підвищенню інтересу до гри, забезпечуючи персоналізований досвід та підтримуючи баланс між складністю ігрових завдань та рівнем підготовки гравців.
- Забезпечення авторизації та аутентифікації користувачів через сучасні протоколи безпеки, такі як OAuth 2.0 та JWT (JSON Web Tokens), що гарантують захищений доступ до системи. Реалізація включає використання багатофакторної автентифікації (MFA), шифрування даних під час передачі та зберігання, а також моніторинг підозрілих активностей для запобігання несанкціонованому доступу.

Ключовими аспектами клієнт-серверної архітектури є:

Модульність. Чіткий поділ на клієнтську і серверну частини спрощує розробку та підтримку системи, дозволяючи кожній частині виконувати специфічні завдання незалежно одна від одної. Такий підхід сприяє легкості масштабування, тестування та оновлення функціоналу без значних змін у загальній структурі додатку.

Реактивність. Використання WebSocket для реальної взаємодії між клієнтами та сервером дозволяє оновлювати дані миттєво та забезпечувати постійний зв’язок у режимі реального часу. Це дозволяє системі реагувати на дії користувачів з мінімальною затримкою, синхронізувати ігрові події для всіх учасників та зменшити навантаження на сервер завдяки оптимізації обміну даними. Реактивність також сприяє покращенню користувацького досвіду за рахунок плавної роботи навіть під час пікових навантажень.

Безпека. Захищені канали зв’язку між клієнтом і сервером (HTTPS, WebSocket Secure) забезпечують шифрування переданої інформації, що мінімізує ризик перехоплення даних або їх зміни під час передачі. Додатково реалізовані механізми багатофакторної автентифікації, перевірка цілісності даних та регулярний моніторинг потенційних загроз. Система підтримує сучасні криптографічні алгоритми, а також активне виявлення та запобігання атакам, включаючи DDoS та MITM. Це гарантує максимальний рівень безпеки для користувачів та стабільність роботи додатку.

Архітектура клієнт-серверної взаємодії забезпечує стабільність роботи додатку завдяки ефективному управлінню ресурсами, оптимізації потоків даних та розподілу навантаження між клієнтською і серверною частинами. Вона дозволяє легко масштабувати функціональність системи шляхом додавання нових модулів чи інтеграції з сучасними сервісами, такими як хмарні обчислення, інструменти машинного навчання або сервіси для аналізу великих даних. Крім того, архітектура підтримує динамічне балансування навантаження в реальному часі, яке базується на аналізі поточних ресурсів та передбаченні пікових моментів завантаження

системи. Це гарантує стабільність і швидкість обробки навіть під час пікових навантажень, забезпечуючи мінімальні затримки у відповідях серверу.

Висока продуктивність досягається завдяки застосуванню таких технологій, як оптимізовані механізми кешування для зменшення навантаження на базу даних, використання CDN для швидкої доставки статичних файлів та горизонтальне масштабування серверів, яке дозволяє системі зберігати свою продуктивність незалежно від зростання кількості користувачів. Додатково інтегруються передові алгоритми обробки запитів, такі як машинне навчання для передбачення навантажень, що дозволяє заздалегідь підготувати необхідні ресурси і уникнути перевантаження. Забезпечує не лише високу ефективність роботи системи, але й підвищує її адаптивність до змінних вимог, роблячи її гнучкою, надійною та здатною відповідати найвищим стандартам продуктивності навіть у довгостроковій перспективі..

2.3 Інтеграція штучного інтелекту через Claude API

Інтеграція штучного інтелекту (ШІ) у веб-додаток через Claude API є ключовим елементом, що значно підвищує функціональність ігрової системи. Claude API надає потужні інструменти для реалізації адаптивного ігрового досвіду, аналізу поведінки користувачів, автоматизації складних процесів і впровадження інтерактивних інновацій. Це забезпечує створення інтуїтивно зрозумілого продукту, здатного не лише адаптуватися до потреб користувачів, але й передбачати їхні очікування за допомогою прогнозувальних моделей. Застосування Claude API дає змогу створювати інтерактивне середовище, яке в реальному часі аналізує багатофакторні дані, такі як ігрові метрики, активність, індивідуальні уподобання користувачів, і на основі цього генерує оптимізовані сценарії гри. Це сприяє формуванню не лише високого рівня залучення, але й постійного розвитку користувацького досвіду, підвищуючи задоволення від гри та стимулюючи глибоку емоційну прив'язаність до платформи. Крім того, система підтримує багаторівневу адаптацію контенту, яка дозволяє враховувати зміни в

поведінці гравців та динамічно налаштовувати ігрові механіки відповідно до змінних умов.

```
// Приклад інтеграції Claude API у Node.js за допомогою axios
const axios = require('axios');
const API_KEY = 'your_claude_api_key';
const BASE_URL = 'https://api.claude.ai';
// Функція для надсилання запиту до Claude API
async function analyzeUserData(userData) {
    try {
        const response = await axios.post(`${BASE_URL}/analyze`,
{
            data: userData,
        }, {
            headers: {
                'Authorization': `Bearer ${API_KEY}`,
                'Content-Type': 'application/json',
            },
        });
        return response.data; // Повернення результату аналізу
    } catch (error) {
        console.error('Помилка інтеграції з Claude API:', error);
        throw error;
    }
}
// Використання функції для аналізу даних користувача
const userData = {
    metrics: {
        activity: 85,
        successRate: 78,
        engagement: 90,
    },
    preferences: ['adventure', 'strategy'],
};
```

```
analyzeUserData(userData)
    .then(result => {
        console.log('Результати аналізу:', result);
    })
    .catch(error => {
        console.error('Помилка під час аналізу даних:', error);
    });
```

Claude API використовується для аналізу даних, зібраних під час взаємодії користувачів із системою. Цей аналіз охоплює різноманітні аспекти, включаючи частоту дій гравців, час виконання завдань, рівень взаємодії з ключовими елементами гри, а також їхні успіхи чи труднощі на різних етапах. Система використовує ці дані для формування багаторівневих аналітичних моделей, які дозволяють прогнозувати потреби гравців з високою точністю. На основі отриманих висновків система динамічно адаптує рівень складності завдань, створює персоналізовані ігрові сценарії, генерує релевантні рекомендації для покращення досвіду користувача та оптимізує порядок подання ігрового контенту. Завдяки цьому підходу досягається не лише високий рівень залучення, але й створюється довготривалий інтерес до гри, формуючи стійку емоційну прив'язаність гравців до платформи та стимулюючи їх повернення.

```
# Приклад інтеграції Claude API у Python
import requests
API_KEY = 'your_claude_api_key'
BASE_URL = 'https://api.claude.ai'
def analyze_user_data(user_data):
    try:
        headers = {
            'Authorization': f'Bearer {API_KEY}',
            'Content-Type': 'application/json',
        }
```

```
            response = requests.post(f'{BASE_URL}/analyze',
json={'data': user_data}, headers=headers)
            response.raise_for_status()
            return response.json()
        except requests.exceptions.RequestException as e:
            print(f'Error integrating with Claude API: {e}')
            return None
    # Приклад даних користувача
    user_data = {
        'metrics': {
            'activity': 85,
            'successRate': 78,
            'engagement': 90,
        },
        'preferences': ['adventure', 'strategy'],
    }
    result = analyze_user_data(user_data)
    if result:
        print('Результати аналізу:', result)
    else:
        print('Не вдалося отримати результати аналізу.')
```

Однією з основних переваг інтеграції Claude API є можливість роботи з природною мовою. Використання ШІ дозволяє системі аналізувати текстові запити, генерувати відповіді в реальному часі та підтримувати багатогранний динамічний діалог між користувачем і додатком. Це охоплює не лише текстову комунікацію, але й складні інтерактивні сценарії, які забезпечують адаптивне налаштування ігрових завдань відповідно до індивідуальних потреб користувача. Claude API дозволяє інтегрувати голосові команди, надаючи користувачам нові рівні взаємодії, включаючи можливість керувати ігровими процесами голосом або отримувати аудіопідказки в реальному часі. Гравці можуть ставити запитання, отримувати розширені пояснення та персоналізовані рекомендації, які базуються

на їхніх діях у грі. Інтерактивний інтерфейс також дозволяє користувачам отримувати підтримку, виявляти свої сильні сторони та розвивати ігрові навички, що сприяє глибшому зануренню в гру. Такі можливості значно підвищують якість користувацького досвіду, роблячи взаємодію більш природною, інтуїтивною та ефективною.

Claude API також підтримує механізми навчання моделей, які постійно вдосконалюються на основі нових даних, зокрема завдяки використанню нейронних мереж для аналізу складних взаємозв'язків у поведінці користувачів. Інтеграція API дозволяє системі прогнозувати не лише дії окремих гравців, але й тренди в ігровій спільноті загалом, що є важливим для підтримки довгострокової релевантності контенту. Система використовує ці дані для створення адаптивних стратегій, автоматичного оновлення ігрових механік і покращення користувацького досвіду через впровадження персоналізованих рекомендацій у реальному часі. Claude API здатен ефективно обробляти надвеликі обсяги даних, аналізуючи їх у реальному часі, що дозволяє мінімізувати затримки у взаємодії та забезпечувати плавність ігрового процесу. Інтеграція API має високий потенціал масштабованості, дозволяючи одночасно обробляти мільйони запитів і гарантувати стабільну роботу навіть у проєктах із критично високими навантаженнями, що підвищує надійність системи та довіру користувачів.

```
// Приклад інтеграції Claude API у Java
import java.net.HttpURLConnection;
import java.net.URL;
import java.io.OutputStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class ClaudeAPIIntegration {
    private static final String API_KEY = "your_claude_api_key";
    private static final String BASE_URL = 
"https://api.claude.ai/analyze";
    public static String analyzeUserData(String userData) {
```

```
            try {
                URL url = new URL(BASE_URL);
                HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
                connection.setRequestMethod("POST");
                connection.setRequestProperty("Authorization",
"Bearer " + API_KEY);
                connection.setRequestProperty("Content-Type",
"application/json");
                connection.setDoOutput(true);
                try (OutputStream os = connection.getOutputStream())
{
                    os.write(userData.getBytes("UTF-8"));
                    os.flush();
                }
                int responseCode = connection.getResponseCode();
                if (responseCode == HttpURLConnection.HTTP_OK) {
                    BufferedReader in = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
                    String inputLine;
                    StringBuilder response = new StringBuilder();
                    while ((inputLine = in.readLine()) != null) {
                        response.append(inputLine);
                    }
                    in.close();
                    return response.toString();
                } else {
                    return "Error: " + responseCode;
                }
            } catch (Exception e) {
                return "Exception: " + e.getMessage();
            }
        }
        public static void main(String[] args) {
```

```
            String                          userData                          =
"{\"metrics\":{\"activity\":85,\"successRate\":78,\"engagement\":90}
,\"preferences\":[\"adventure\",\"strategy\"]}";
            String result = analyzeUserData(userData);
            System.out.println("Результати аналізу: " + result);
        }
    }
```

Інтеграція Claude API у веб-додаток забезпечує значні конкурентні переваги, дозволяючи створити інноваційний продукт із високим рівнем персоналізації, адаптивності та багатовимірної взаємодії. Завдяки глибокій інтеграції з механізмами штучного інтелекту, система здатна автоматично навчатися на базі нових даних, вдосконалюючи свої алгоритми для підтримки актуальності, відповідності сучасним вимогам і врахування змінних потреб користувачів. Claude API не лише оптимізує ігрові процеси, але й активно сприяє розвитку платформи, дозволяючи впроваджувати нові ігрові механіки, автоматизовані рекомендації та прогнозування поведінки користувачів. Це забезпечує платформі не тільки гнучкість, але й стійкість до швидких змін на ринку технологій, сприяючи довгостроковій конкурентоспроможності, надійності та розширенню функціональних можливостей..

2.4 Висновки по розділу

У другому розділі було проведено детальне проєктування веб-додатку пізнавальної гри з інтеграцією сучасних технологій та інструментів. Розглянуто архітектурні рішення, які забезпечують високу продуктивність, гнучкість і масштабованість системи. Клієнт-серверна архітектура дозволяє ефективно розподіляти функціонал, забезпечуючи стабільність роботи навіть при високому навантаженні.

Інтеграція Claude API відкриває нові можливості для персоналізації та адаптивності гри, використовуючи штучний інтелект для аналізу поведінки

користувачів та створення унікального досвіду. Наведені приклади коду демонструють практичну реалізацію таких технологій, що підвищує практичну цінність дослідження.

Під час розробки концептуальної моделі враховувалися як потреби користувачів, так і сучасні тенденції в розробці програмного забезпечення. Використання інноваційних підходів дозволяє створити продукт, що відповідає високим стандартам якості, забезпечуючи довготривалу конкурентоспроможність. Таким чином, проєктування веб-додатку демонструє значний потенціал для подальшого розвитку та адаптації до мінливих умов ринку.

3 МЕТОДИ РЕАЛІЗАЦІЇ ТА ЗАБЕЗПЕЧЕННЯ ЕФЕКТИВНОСТІ РОБОТИ ДОДАТКУ

3.1 Розробка клієнтської частини на Angular: основні принципи

Angular є потужним фреймворком для розробки динамічних односторінкових веб-додатків (SPA), який забезпечує високий рівень продуктивності, масштабованості та гнучкість у реалізації користувацького інтерфейсу. Завдяки використанню сучасних технологій, таких як двосторонній зв'язок даних, модульна архітектура та реактивне програмування через RxJS, Angular дозволяє створювати інтерактивні та адаптивні веб-додатки, що відповідають сучасним вимогам користувачів. Angular також підтримує інтеграцію з іншими фреймворками та бібліотеками, забезпечуючи зручну розробку складних додатків, включаючи ті, що вимагають високої продуктивності та великої кількості компонентів.

Розробка клієнтської частини на Angular сприяє побудові плавного, інтуїтивного користувацького досвіду, забезпечуючи оптимальну продуктивність навіть за умов високого навантаження. Angular забезпечує високу інтерактивність завдяки двосторонньому зв'язку даних, підтримує динамічне оновлення контенту без перезавантаження сторінок, що створює комфорт для користувачів. Його можливості, такі як AOT-компіляція для зменшення часу завантаження, інтеграція з хмарними сервісами через REST або GraphQL, а також підтримка прогресивних веб-додатків (PWA), відкривають нові горизонти для побудови масштабованих, адаптивних і стабільних рішень. Використання цих інструментів гарантує швидкість роботи навіть у великих проєктах, забезпечуючи відповідність сучасним стандартам розробки.

Основні принципи розробки клієнтської частини на Angular включають

Модульність. Angular підтримує модульну архітектуру, яка дозволяє структурувати код у вигляді компонентів, сервісів і модулів. Це спрощує підтримку, розширення та тестування додатку. Наприклад, створення базового компонента може виглядати так:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: `<h1>Вітаємо у нашому додатку!</h1>`,
  styles: [`h1 { color: blue; }`]
})
export class AppComponent {
  title = 'Мій Angular додаток';
}
```

Двосторонній зв'язок даних (Two-Way Data Binding). Цей механізм забезпечує синхронізацію між моделлю даних і користувацьким інтерфейсом у реальному часі, що значно підвищує інтерактивність додатку. Наприклад:

```
<input [(ngModel)]="username" placeholder="Введіть ім'я">
<p>Привіт, {{ username }}!</p>
import { Component } from '@angular/core';
@Component({
  selector: 'app-user-input',
  templateUrl: './user-input.component.html',
})
export class UserInputComponent {
  username: string = '';
}
```

Шаблони та директиви. Angular надає інструменти для створення динамічного інтерфейсу за допомогою шаблонів HTML та власних директив, що дозволяє реалізувати адаптивний дизайн і інтерактивні елементи. Наприклад, використання директиви *ngIf:

```
<div *ngIf="isLoggedIn">Ласкаво просимо, користувачу!</div>
```

```
<button (click)="toggleLogin()">Перемкнути статус входу</button>
import { Component } from '@angular/core';
@Component({
  selector: 'app-login-toggle',
  templateUrl: './login-toggle.component.html',
})
export class LoginToggleComponent {
  isLoggedIn: boolean = false;

  toggleLogin() {
    this.isLoggedIn = !this.isLoggedIn;
  }
}
```

Dependency Injection (DI). Використання DI сприяє покращенню архітектури додатку, забезпечуючи зручне управління залежностями та розширюваність системи.

Оптимізація продуктивності. Angular підтримує техніки Lazy Loading та Ahead-of-Time Compilation (AOT), що дозволяє мінімізувати час завантаження додатку та підвищити його продуктивність. Було детально розглянуто основні принципи використання Angular для розробки клієнтської частини веб-додатку. Реалізація цих принципів, включаючи модульність, двосторонній зв'язок даних, використання директив і шаблонів, а також оптимізацію продуктивності, сприяє створенню стабільного, інтерактивного та ефективного користувацького досвіду.

Застосування Angular у проєктах дозволяє використовувати такі передові техніки, як Lazy Loading, що забезпечує завантаження лише необхідних модулів, та AOT-компіляцію, яка мінімізує затримки під час виконання додатку. Наприклад:

```
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'dashboard', loadChildren: () => import('./dashboard/dashboard.module').then(m => m.DashboardModule) }
```

```
];
```

Це сприяє підвищенню продуктивності навіть у великих додатках із великою кількістю компонентів.

Завдяки інтеграції Dependency Injection, Angular полегшує тестування і підтримку сервісів, що демонструється у простому прикладі:

```
@Injectable({ providedIn: 'root' })
export class UserService {
  private users = ['John', 'Jane', 'Doe'];
  getUsers() {
    return this.users;
  }
}
```

У сукупності ці підходи демонструють практичний підхід до інтеграції сучасних стандартів веб-розробки, забезпечуючи довгострокову стабільність, високу продуктивність та зручність роботи з клієнтською частиною. Angular дозволяє поєднувати інноваційні підходи до організації коду, такі як використання декораторів, наприклад @Component і @Injectable, для спрощення управління залежностями та компонування додатку.

Окрім цього, інтеграція механізмів реактивного програмування через бібліотеку RxJS дає змогу створювати асинхронні потоки даних, оптимізуючи взаємодію з API. Наприклад:

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
@Injectable({ providedIn: 'root' })
export class DataService {
  constructor(private http: HttpClient) {}
```

```
  getData(): Observable<any> {
    return this.http.get('https://api.example.com/data');
  }
}
```

Такі інновації забезпечують не лише продуктивність, але й високий рівень інтерактивності та зручності користування. Angular надає потужний інструментарій для вирішення складних завдань сучасних веб-додатків, забезпечуючи адаптивність інтерфейсу, інтеграцію зі сторонніми API та підтримку масштабованості. Його можливості, такі як використання модульного підходу, реактивного програмування та механізмів Lazy Loading, дозволяють ефективно працювати з великими обсягами даних та забезпечувати швидкий відгук користувачам. Angular також підтримує впровадження прогресивних веб-додатків (PWA), які забезпечують офлайн-доступ до контенту та покращують продуктивність навіть на слабких пристроях. Інтеграція з бібліотеками для візуалізації даних, такими як D3.js чи Chart.js, дозволяє створювати інтерактивні графіки та діаграми, збагачуючи досвід користувачів.

Angular забезпечує підтримку серверного рендерингу через Angular Universal, що покращує SEO-оптимізацію та швидкість завантаження сторінок. Використання CLI (Command Line Interface) спрощує ініціалізацію проєктів, автоматизацію рутинних завдань та управління залежностями. Така гнучкість і багатофункціональність робить Angular ідеальним вибором для розробки як корпоративних систем, так і клієнтоорієнтованих продуктів.

3.2 Серверна частина з NestJS: ключові особливості

NestJS є сучасним фреймворком для розробки серверної частини, що базується на TypeScript і Node.js. Він поєднує в собі переваги об'єктно-орієнтованого програмування, функціонального підходу та реактивного програмування, що дозволяє створювати високопродуктивні, масштабовані та добре структуровані веб-додатки. Завдяки модульній архітектурі,

автоматизованому управлінню залежностями, а також підтримці сучасних стандартів веб-розробки, NestJS забезпечує розробникам інструменти для створення стабільних і ефективних рішень. NestJS підтримує передові підходи до побудови серверних додатків, такі як мікросервісна архітектура, яка дозволяє створювати розподілені системи з високою доступністю. Завдяки інтеграції з сучасними інструментами, такими як Kubernetes і Docker, фреймворк спрощує розгортання та управління додатками у хмарних середовищах. Крім того, NestJS активно використовує TypeScript для підвищення безпеки коду та зменшення кількості помилок під час розробки. Його гнучкість і потужність дозволяють легко адаптуватися до сучасних вимог, забезпечуючи високу якість і надійність у масштабних проєктах.

Фреймворк пропонує інструменти для побудови сервіс-орієнтованої архітектури (SOA) та мікросервісів, дозволяючи легко інтегруватися з іншими системами за допомогою API та підтримки стандартів, таких як gRPC або GraphQL. NestJS також забезпечує можливості для побудови асинхронних сервісів, які можуть взаємодіяти з чергами повідомлень, такими як RabbitMQ чи Kafka, для забезпечення ефективності в обробці великих обсягів даних. Крім того, він надає інструменти для централізованого логування, моніторингу продуктивності через інтеграцію з Prometheus або Elastic Stack, і впровадження найкращих практик безпеки, таких як використання шифрування даних та захисту від атак на основі ролей. Ці функції є критично важливими для сучасних високонавантажених додатків, що працюють у масштабованих середовищах.

У поєднанні з його гнучкістю, потужністю та підтримкою найкращих практик, NestJS стає оптимальним вибором для складних і вимогливих проєктів. Він забезпечує стабільність, адаптивність до сучасних технологічних викликів та ефективну роботу навіть у високонавантажених середовищах. Завдяки вбудованим можливостям інтеграції з іншими технологіями, наприклад, мікросервісними архітектурами або хмарними обчисленнями через AWS чи Azure, NestJS дозволяє створювати надійні системи з високою доступністю. Його багатофункціональність

включає підтримку різних протоколів комунікації, таких як HTTP/2, WebSocket та gRPC, що дозволяє масштабувати додатки та забезпечувати швидкість обробки запитів. Фреймворк також надає вбудовані можливості для тестування додатків, зокрема підтримку інтеграційного та юніт-тестування, що полегшує забезпечення якості на кожному етапі розробки. Завдяки таким особливостям NestJS відповідає найвищим стандартам розробки, гарантуючи успішну реалізацію як невеликих, так і масштабних проєктів.

Модульна архітектура. NestJS організовує код у вигляді модулів, які можуть бути незалежними або взаємопов'язаними. Така структура сприяє логічному розподілу функціоналу, забезпечує легке тестування та масштабування додатку. Наприклад:

```
import { Module } from '@nestjs/common';
import { UsersController } from './users.controller';
import { UsersService } from './users.service';
@Module({
  controllers: [UsersController],
  providers: [UsersService],
})
export class UsersModule {}
```

Використання Dependency Injection (DI). Dependency Injection є ключовим елементом архітектури NestJS, який дозволяє ефективно управляти залежностями між компонентами, сприяючи модульності та повторному використанню коду. Завдяки DI розробники можуть легко створювати й тестувати окремі компоненти, оскільки всі залежності передаються через конструктор або інші механізми. Наприклад:

```
import { Injectable } from '@nestjs/common';
@Injectable()
export class UsersService {
```

```
  private users = ['Alice', 'Bob'];
  getUsers(): string[] {
    return this.users;
  }
}
```

Реалізація RESTful API. NestJS спрощує створення RESTful-інтерфейсів завдяки зручним декораторам для маршрутів. Ці декоратори, такі як @Controller, @Get, @Post, дозволяють швидко та ефективно описувати маршрути та відповідні їм методи обробки запитів. Завдяки цьому структура коду стає інтуїтивно зрозумілою, а додаток легко підтримувати та масштабувати. Крім того, NestJS підтримує додаткові функції, такі як параметри маршрутів, валідація запитів і автоматична генерація документації через інтеграцію з OpenAPI (Swagger).

```
import { Controller, Get } from '@nestjs/common';
@Controller('users')
export class UsersController {
  @Get()
  findAll(): string {
    return 'This action returns all users';
  }
}
```

Підтримка WebSocket. Для реалізації реального часу NestJS надає інтеграцію з WebSocket, яка дозволяє створювати інтерактивні додатки з низькою затримкою. Ця технологія є ключовою для чатів, систем сповіщень, багатокористувацьких ігор та інших додатків, де потрібна миттєва взаємодія. Завдяки простим у використанні декораторам, таким як @WebSocketGateway та @SubscribeMessage, розробники можуть легко налаштувати маршрути обробки повідомлень і створювати адаптивні рішення для роботи в режимі реального часу. Крім того, NestJS забезпечує підтримку масштабування WebSocket через адаптери,

такі як Redis, що дозволяє реалізовувати складні архітектури з розподіленими вузлами.

```
import { WebSocketGateway, SubscribeMessage, MessageBody } from
'@nestjs/websockets';
@WebSocketGateway()
export class ChatGateway {
  @SubscribeMessage('message')
  handleMessage(@MessageBody() data: string): string {
    return `Received: ${data}`;
  }
}
```

Масштабованість та інтеграція. NestJS підтримує інтеграцію з різними базами даних (наприклад, MongoDB, PostgreSQL) через ORM (TypeORM, Prisma), що дозволяє ефективно працювати з даними в складних додатках. Крім того, фреймворк забезпечує можливість горизонтального масштабування серверів, що робить його ідеальним для розподілених систем. Інтеграція з системами кешування, такими як Redis або Memcached, додає швидкості при обробці великої кількості запитів. Завдяки підтримці хмарних платформ, таких як AWS, Google Cloud, або Azure, NestJS спрощує побудову хмарних архітектур, що відповідають сучасним стандартам.

Безпека. NestJS дозволяє легко впроваджувати аутентифікацію та авторизацію через модулі, такі як Passport.js, що забезпечують гнучкий підхід до управління доступом користувачів. Підтримка таких методів аутентифікації, як JWT (JSON Web Tokens), OAuth2 і стратегій на основі сесій, дозволяє адаптуватися до різних вимог додатку. Крім того, NestJS підтримує захищені протоколи передачі даних, зокрема HTTPS і TLS, а також забезпечує можливість реалізації захисту від атак, таких як CSRF (Cross-Site Request Forgery) і XSS (Cross-Site Scripting). Це

робить фреймворк надійним вибором для розробки додатків із високими вимогами до безпеки.

NestJS забезпечує стабільність, гнучкість і ефективність у розробці серверної частини додатків. Його архітектура сприяє швидкій адаптації до змінних вимог, дозволяє легко інтегруватися з сучасними технологіями та підтримувати високу продуктивність навіть у високонавантажених середовищах. Завдяки використанню модульної структури, автоматизованому управлінню залежностями та підтримці передових інструментів, таких як мікросервісна архітектура, хмарні платформи, протоколи WebSocket і гнучкі системи кешування, NestJS дозволяє створювати адаптивні, масштабовані рішення. Фреймворк також спрощує розробку через інтеграцію з такими інструментами, як OpenAPI (Swagger) для автоматизації документації, що зменшує час на налаштування і тестування додатків, забезпечуючи високу якість кінцевого продукту.

3.3 Організація інтерактивності через WebSocket: основи та підходи

WebSocket є потужною технологією, яка дозволяє створювати інтерактивні додатки в реальному часі з низькими затримками. Ця технологія забезпечує двонаправлений зв'язок між клієнтом і сервером, що відкриває широкі можливості для розробки додатків у різних галузях, таких як чати, багатокористувацькі ігри, біржові платформи, системи моніторингу даних, онлайн-конференції, медичні системи, автоматизація управління пристроями IoT, системи обміну повідомленнями, платформи спільного редагування документів, стрімінгові сервіси та інші рішення, що потребують високої інтерактивності, точності передачі даних та мінімальних затримок. У цьому підрозділі розглянемо основи використання WebSocket та підходи до його інтеграції в додаток.

Основи роботи WebSocket WebSocket забезпечує постійне з'єднання між клієнтом і сервером, дозволяючи обмінюватися даними без необхідності повторного встановлення HTTP-запитів. Це значно знижує затримки, підвищує продуктивність системи та забезпечує можливість обробки даних у реальному часі.

Завдяки підтримці двонаправленої передачі даних WebSocket дозволяє забезпечити високу інтерактивність додатків. Вона є критично важливою для багатокористувацьких середовищ, таких як онлайн-ігри, торгові платформи чи стрімінгові сервіси, а також для систем моніторингу реального часу, платформ фінансових транзакцій і інтерактивних навчальних інструментів. WebSocket інтегрується із сучасними технологіями, такими як обробка великих даних (Big Data) і машинне навчання (ML), що дозволяє створювати інноваційні рішення з високим рівнем адаптивності до вимог користувачів.

Типовий робочий процес із WebSocket:

- Клієнт надсилає запит на встановлення WebSocket-з'єднання.
- Сервер приймає запит і встановлює двонаправлене з'єднання.
- Дані можуть передаватися в обидва боки в режимі реального часу.

Інтеграція WebSocket у NestJS NestJS надає потужний інструментарій для роботи з WebSocket завдяки модулю @nestjs/websockets та підтримці адаптерів, які розширюють можливості масштабування та інтеграції. Основні кроки для організації інтерактивності:

Створення WebSocket-шлюзу:

```
import { WebSocketGateway, SubscribeMessage, MessageBody, OnGatewayConnection, OnGatewayDisconnect } from '@nestjs/websockets';
import { Logger } from '@nestjs/common';
@WebSocketGateway()
export class ChatGateway implements OnGatewayConnection, OnGatewayDisconnect {
  private readonly logger = new Logger(ChatGateway.name);
  handleConnection(client: any) {
    this.logger.log(`Client connected: ${client.id}`);
  }
  handleDisconnect(client: any) {
    this.logger.log(`Client disconnected: ${client.id}`);
  }
```

```
  @SubscribeMessage('message')
  handleMessage(@MessageBody() data: string): string {
    this.logger.log(`Message received: ${data}`);
    return `Echo: ${data}`;
  }
}
```

Даний код демонструє створення WebSocket-шлюзу, який відповідає за обробку підключень, відключень, а також надсилання й отримання повідомлень клієнтів. Крім того, його можна розширити для підтримки додаткових функцій, таких як автентифікація, авторизація та логування даних для аналітики або моніторингу в реальному часі.

Реалізація обробки подій: Використання декоратора @SubscribeMessage дозволяє налаштовувати обробники для різних типів повідомлень. Це забезпечує гнучкість і дозволяє легко впроваджувати обробку складних сценаріїв взаємодії, наприклад підтвердження доставлення повідомлень або оновлення стану сесій у реальному часі. Для підтримки широкомовних повідомлень:

```
@WebSocketGateway()
export class NotificationGateway {
  private clients = new Set<any>();
  handleConnection(client: any) {
    this.clients.add(client);
  }
  handleDisconnect(client: any) {
    this.clients.delete(client);
  }
  broadcastMessage(message: string) {
    this.clients.forEach(client => {
      client.send(message);
```

```
    });
  }
}
```

Масштабування через адаптери: Для підтримки багатокористувацького режиму в розподілених середовищах можна використовувати адаптери, наприклад Redis:

```
import { IoAdapter } from '@nestjs/platform-socket.io';
import * as redisAdapter from 'socket.io-redis';
export class RedisIoAdapter extends IoAdapter {
  createIOServer(port: number, options?: any): any {
    const server = super.createIOServer(port, options);
    server.adapter(redisAdapter({ host: 'localhost', port: 6379
}));
    return server;
  }
}
// Додатково можна реалізувати логіку для моніторингу стану
серверів та оптимізації підключень:
server.on('connection', (socket) => {
  console.log(`New client connected: ${socket.id}`);
  socket.on('disconnect', () => {
    console.log(`Client disconnected: ${socket.id}`);
  });
});
// Підтримка масштабування за допомогою кластеризації Node.js:
import cluster from 'cluster';
import os from 'os';
if (cluster.isMaster) {
  const numCPUs = os.cpus().length;
  for (let i = 0; i < numCPUs; i++) {
    cluster.fork();
  }
```

```
  cluster.on('exit', (worker, code, signal) => {
    console.log(`Worker ${worker.process.pid} died`);
  });
} else {
  // Код сервера WebSocket
}
```

Переваги використання WebSocket

- Низькі затримки. Підходить для систем, які вимагають обробки даних у реальному часі.
- Ефективність. Постійне з'єднання знижує кількість накладних витрат на встановлення нових HTTP-запитів.
- Масштабованість. Завдяки підтримці адаптерів, таких як Redis, WebSocket може обслуговувати велику кількість одночасних клієнтів.

Впровадження WebSocket у додатки на основі NestJS дозволяє створювати високопродуктивні та інтерактивні рішення, які відповідають сучасним вимогам. Завдяки гнучкості та підтримці масштабованих архітектур, WebSocket забезпечує швидкий обмін даними навіть у середовищах із високою кількістю одночасних підключень. Його переваги дозволяють ефективно інтегрувати функції реального часу в додатки, включаючи інтелектуальні системи моніторингу, автоматичну передачу великих обсягів даних і підтримку потокових трансляцій. Крім того, WebSocket ідеально підходить для застосунків, де потрібна синхронізація віртуальних робочих середовищ, таких як платформи для віддаленої співпраці, інтерактивні ігрові хаби та системи навчання. Його інтеграція зі сторонніми сервісами, такими як хмарні обчислення або платформи аналізу даних, підвищує ефективність та розширює можливості масштабування.

3.4 Висновки по розділу

У цьому розділі було розглянуто методи реалізації клієнтської та серверної частини веб-додатку, інтеграцію сучасних технологій та забезпечення ефективності його роботи. Основні аспекти, які було висвітлено:

1. Використання Angular для створення клієнтської частини, що забезпечує інтуїтивний користувацький досвід, високу продуктивність і адаптивність додатку.
2. Застосування NestJS для побудови серверної частини, яка характеризується модульністю, масштабованістю та інтеграцією сучасних технологій.
3. Впровадження WebSocket для організації інтерактивності в реальному часі, що є критично важливим для багатокористувацьких систем і платформ, які потребують низьких затримок та високої надійності.

Реалізація цих технологій у рамках одного додатку дозволяє забезпечити його масштабованість, стабільність та відповідність сучасним стандартам розробки. Використання таких інструментів, як Redis для масштабування, інтеграція з хмарними платформами та адаптація до високих навантажень, відкриває широкі перспективи для розвитку та вдосконалення веб-додатків у майбутньому..

4 ШЛЯХИ УДОСКОНАЛЕННЯ ТА ПЕРСПЕКТИВИ РОЗВИТКУ ВЕБ-ДОДАТКУ

4.1 Оптимізація швидкодії клієнтської і серверної частин

Оптимізація швидкодії є важливим етапом у розвитку веб-додатків, оскільки забезпечує покращення досвіду користувачів і підвищення продуктивності системи. Цей процес включає як технічні аспекти, такі як оптимізація коду, архітектури та баз даних, так і використання сучасних інструментів для моніторингу та аналізу. Основними завданнями оптимізації є зменшення затримок, прискорення обробки даних і забезпечення стабільності навіть за умов високих навантажень. У цьому підрозділі детально розглянемо ключові підходи до вдосконалення клієнтської та серверної частин, інтеграцію передових практик програмування, впровадження механізмів кешування та використання асинхронних технологій для обробки великих обсягів даних у реальному часі. Розглянемо також оптимізацію інфраструктури, включаючи масштабування серверів і інтеграцію з хмарними платформами.

Оптимізація клієнтської частини

Зменшення розміру ресурсів: Використання методів, таких як мініфікація JavaScript, CSS та HTML, а також стиснення зображень за допомогою сучасних форматів (WebP, AVIF), значно зменшує час завантаження сторінок. Наприклад, можна використовувати спеціалізовані інструменти, такі як Terser для JavaScript, PostCSS для CSS або ImageMagick для оптимізації зображень. Впровадження таких підходів дозволяє знизити об'єм переданих даних та зменшити час відгуку сервера, що позитивно впливає на швидкодію додатку.

```
// Приклад мініфікації JavaScript за допомогою Terser
const Terser = require('terser');
const jsCode = `function helloWorld() { console.log("Hello World"); }`;
const minifiedCode = Terser.minify(jsCode).code;
```

```
console.log(minifiedCode);
```

Впровадження механізмів кешування: Браузерне кешування дозволяє повторно використовувати завантажені раніше ресурси, скорочуючи час завантаження нових сторінок. Для серверного кешування можна використовувати такі підходи, як кешування результатів обчислень або запитів до бази даних у Redis або Memcached. Це значно зменшує навантаження на сервер і прискорює час відповіді.

```
// Приклад реалізації кешування запиту до бази даних
const getCachedData = async (key, queryFunc) => {
  const cached = await redisClient.get(key);
  if (cached) {
    return JSON.parse(cached);
  }
  const data = await queryFunc();
  redisClient.set(key, JSON.stringify(data), 'EX', 3600);
  return data;
};
const fetchData = async () => {
  return await getCachedData('user:123', async () => {
    return db.query('SELECT * FROM users WHERE id = 123');
  });
};
// HTTP-заголовок для кешування
response.setHeader('Cache-Control', 'public, max-age=31536000');
```

Lazy Loading: Завантаження лише тих ресурсів, які потрібні для поточного відображення, значно підвищує швидкодію додатку. Цей підхід дозволяє зменшити початковий час завантаження сторінки та зменшити споживання ресурсів. Lazy Loading активно використовується в фреймворках, таких як Angular та React. У Angular, наприклад, модулі можуть завантажуватися динамічно:

```
// Angular: Приклад Lazy Loading для маршруту
const routes: Routes = [
  { path: 'dashboard', loadChildren: () => import('./dashboard/dashboard.module').then(m => m.DashboardModule) },
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

Також можна реалізувати завантаження зображень або інших мультимедійних ресурсів тільки тоді, коли вони з’являються в полі зору користувача:

```
// Lazy Loading зображень у JavaScript
const images = document.querySelectorAll('img[data-src]');
const loadImage = (image) => {
  image.src = image.dataset.src;
};
const observer = new IntersectionObserver((entries, observer) => {
  entries.forEach(entry => {
    if (entry.isIntersecting) {
      loadImage(entry.target);
      observer.unobserve(entry.target);
    }
  });
});
images.forEach(image => observer.observe(image));
```

Ці підходи значно знижують навантаження на клієнтську частину, дозволяючи зосередити ресурси на виконанні критично важливих завдань, таких як обробка інтерактивних елементів або складних анімацій. Вони також покращують взаємодію з користувачем, забезпечуючи плавність роботи додатку навіть при обробці великих обсягів даних чи використанні мультимедійного контенту.

```
// Приклад Lazy Loading у React
const LazyComponent = React.lazy(() => import('./LazyComponent'));
function App() {
  return (
    <React.Suspense fallback={<div>Loading...</div>}>
      <LazyComponent />
    </React.Suspense>
  );
}
```

Використання CDN: Мережі доставки контенту (Content Delivery Networks) дозволяють швидше доставляти ресурси користувачам завдяки географічно розподіленим серверам. Це зменшує затримки, забезпечуючи оптимальну швидкодію навіть для користувачів у віддалених регіонах. Наприклад, використання популярних сервісів, таких як Cloudflare, Akamai або AWS CloudFront, дозволяє автоматично кешувати ресурси на серверах CDN і знижувати навантаження на основний сервер. Крім того, інтеграція з CDN підтримує функції захисту від DDoS-атак, забезпечуючи не лише швидкість, але й безпеку додатка. Для динамічного контенту можна налаштувати гібридне кешування, коли часто використовувані дані зберігаються у найближчих до користувача вузлах мережі.

Оптимізація серверної частини

Налаштування бази даних: Оптимізація запитів до бази даних включає використання індексів, зменшення кількості JOIN-операцій та впровадження партиціонування таблиць для роботи з великими наборами даних. Використання

ORM (Object-Relational Mapping), таких як Sequelize або TypeORM, допомагає забезпечити ефективне управління транзакціями та мінімізувати складність запитів. Додатково, моніторинг продуктивності за допомогою інструментів, таких як Query Monitor або EXPLAIN у SQL, дозволяє швидко виявляти та виправляти вузькі місця.

```
-- Додавання індексу для оптимізації запиту
CREATE INDEX idx_users_name ON users (name);
```

Кешування на рівні сервера: Використання систем кешування, таких як Redis або Memcached, дозволяє зменшити навантаження на базу даних і прискорити обробку запитів. Наприклад, Redis може використовуватися для зберігання даних про стан сесій користувачів або для кешування результатів складних запитів:

```
// Приклад використання Redis для кешування запитів
const redis = require('redis');
const client = redis.createClient();
// Функція для отримання даних із кешу або бази
async function getDataWithCache(key, queryFunction) {
  const cachedData = await client.getAsync(key);
  if (cachedData) {
    return JSON.parse(cachedData);
  }
  const data = await queryFunction();
  await client.setAsync(key, JSON.stringify(data), 'EX', 3600);
// Термін дії - 1 година
  return data;
}
// Використання функції
getDataWithCache('user:123', () => db.query('SELECT * FROM users
WHERE id = 123'))
  .then(data => console.log(data));
```

Крім того, Redis підтримує інструменти для масштабування, наприклад, через кластеризацію та реплікацію даних, що робить його ідеальним рішенням для систем із високою відвідуваністю.

```
// Приклад використання Redis для кешування
const redis = require('redis');
const client = redis.createClient();
client.set('key', 'value', 'EX', 3600);
client.get('key', (err, value) => {
  console.log(value);
});
```

Горизонтальне масштабування: Додавання нових серверів для обробки запитів забезпечує стабільність і продуктивність при зростанні кількості користувачів. Цей підхід дозволяє рівномірно розподіляти навантаження між кількома вузлами, знижуючи ризик перевантаження окремого сервера. Для ефективного горизонтального масштабування можна використовувати балансувальники навантаження, наприклад, NGINX або AWS Elastic Load Balancer. Крім того, хмарні провайдери, такі як AWS, Google Cloud та Azure, підтримують автоматичне масштабування (Auto Scaling), яке дозволяє автоматично додавати або видаляти сервери залежно від поточного навантаження.

```
// Приклад кластеризації в Node.js
const cluster = require('cluster');
const http = require('http');
const numCPUs = require('os').cpus().length;
if (cluster.isMaster) {
  for (let i = 0; i < numCPUs; i++) {
    cluster.fork();
  }
  cluster.on('exit', (worker, code, signal) => {
```

```
    console.log(`Worker ${worker.process.pid} died`);
  });
} else {
  http.createServer((req, res) => {
    res.writeHead(200);
    res.end('Hello World!');
  }).listen(8000);
}
```

Використання асинхронного програмування: Завдяки асинхронним підходам у Node.js можна обробляти велику кількість запитів одночасно, що значно підвищує продуктивність серверної частини. Асинхронність дозволяє уникнути блокувань під час виконання довготривалих операцій, таких як доступ до бази даних чи зовнішніх API. Це забезпечується за допомогою конструкцій async/await, стрімів (Streams) і подій (Events). Наприклад:

```
// Приклад обробки паралельних запитів за допомогою Promise.all
const fetch = require('node-fetch');
async function fetchMultipleData(urls) {
  const requests = urls.map(url => fetch(url).then(res => res.json()));
  const results = await Promise.all(requests);
  console.log(results);
}
fetchMultipleData([
  'https://api.example.com/data1',
  'https://api.example.com/data2',
  'https://api.example.com/data3'
]);
```

Крім того, асинхронність дозволяє реалізувати ефективну обробку стрімів даних:

```
// Приклад обробки стрімів у Node.js
const fs = require('fs');
const readStream = fs.createReadStream('largefile.txt');
const writeStream = fs.createWriteStream('output.txt');
readStream.on('data', chunk => {
  console.log(`Read chunk: ${chunk.length} bytes`);
  writeStream.write(chunk);
});
readStream.on('end', () => {
  console.log('File reading completed.');
  writeStream.end();
});
```

Асинхронний підхід є критично важливим для високонавантажених систем, забезпечуючи одночасну обробку великої кількості запитів із мінімальними затримками.

```
// Асинхронна функція для обробки запиту
const fetch = require('node-fetch');
async function fetchData(url) {
  const response = await fetch(url);
  const data = await response.json();
  console.log(data);
}
fetchData('https://api.example.com/data');
```

Інструменти для моніторингу та оптимізації

Для контролю за швидкодією додатку рекомендується використовувати сучасні інструменти. Для клієнтської частини, такі як Google Lighthouse та WebPageTest, дозволяють оцінювати швидкість завантаження, визначати ресурси, які уповільнюють роботу, та отримувати рекомендації щодо їх оптимізації. Для

серверної частини варто використовувати Prometheus і Grafana, які забезпечують моніторинг у реальному часі, візуалізацію метрик та аналіз продуктивності.

```
// Приклад інтеграції Prometheus у Node.js
const client = require('prom-client');
const collectDefaultMetrics = client.collectDefaultMetrics;
collectDefaultMetrics();
const httpRequestDurationMicroseconds = new client.Histogram({
  name: 'http_request_duration_ms',
  help: 'Duration of HTTP requests in ms',
  labelNames: ['method', 'route', 'status_code'],
  buckets: [50, 100, 200, 300, 400, 500]
});
module.exports = { httpRequestDurationMicroseconds };
```

Використання цих інструментів дозволяє автоматизувати процес виявлення вузьких місць та оптимізувати як клієнтську, так і серверну частини додатку. Для складних систем рекомендується також інтегрувати системи APM (Application Performance Monitoring), такі як New Relic або Datadog, які забезпечують повну видимість продуктивності додатку.

Оптимізація швидкодії клієнтської та серверної частин не тільки покращує досвід користувачів, але й знижує витрати на інфраструктуру, забезпечуючи стабільність, масштабованість і довготривалу стійкість системи до зростаючих вимог. Завдяки впровадженню сучасних підходів до обробки даних, таких як асинхронні процеси, кешування та горизонтальне масштабування, веб-додатки можуть залишатися ефективними навіть у високонавантажених середовищах, відповідаючи найвищим стандартам продуктивності..

4.2 Розширення функціоналу на основі нових можливостей Claude API

Claude API пропонує широкий спектр можливостей для інтеграції штучного інтелекту у веб-додатки, дозволяючи створювати інноваційний функціонал, який

покращує користувацький досвід та розширює потенціал системи. У цьому підрозділі розглянемо основні підходи до розширення функціоналу за допомогою Claude API та практичні приклади його застосування.

Основні можливості Claude API:

- Обробка природної мови (NLP): Claude API дозволяє створювати інтелектуальних чат-ботів, які розуміють контекст користувацьких запитів і можуть генерувати відповідні відповіді.
- Аналіз даних: Інструменти для класифікації, оцінювання тональності тексту та пошуку ключових фраз у великих обсягах даних.
- Машинний переклад: Реалізація багатомовної підтримки для інтерфейсу додатку.
- Побудова рекомендаційних систем: Використання моделей Claude для аналізу поведінки користувачів і надання персоналізованих рекомендацій.

Розпізнавання образів: Аналіз зображень для автоматизації процесів, наприклад, ідентифікація об’єктів чи розпізнавання тексту.

Приклади використання Claude API:

```
// Інтеграція Claude API для обробки тексту
const axios = require('axios');
async function analyzeText(text) {
  const response = await axios.post('https://api.claude.ai/analyze', {
    text: text
  }, {
    headers: {
      'Authorization': 'Bearer YOUR_API_KEY'
    }
  });
  return response.data;
}
```

```
analyzeText("What is the weather like today?").then(result => {
  console.log(result);
});
// Використання Claude API для генерації тексту
async function generateText(prompt) {
  const response = await axios.post('https://api.claude.ai/generate', {
    prompt: prompt
  }, {
    headers: {
      'Authorization': 'Bearer YOUR_API_KEY'
    }
  });
  return response.data.text;
}
generateText("Напиши огляд про застосунок штучного інтелекту.").then(text => {
  console.log(text);
});
```

Розширення функціоналу

Інтелектуальні відповіді: Додаток може використовувати Claude API для надання детальних і контекстно залежних відповідей у чатах.

```
// Приклад інтелектуальних відповідей
async function getSmartReply(inputMessage) {
  const response = await axios.post('https://api.claude.ai/smart-reply', {
    message: inputMessage
  }, {
    headers: {
      'Authorization': 'Bearer YOUR_API_KEY'
    }
  });
```

```
  return response.data.reply;
}
getSmartReply("How can I reset my password?").then(reply => {
  console.log(reply);
});
```

Розпізнавання емоцій: Використання API для аналізу тональності тексту дозволяє адаптувати інтерфейс додатку до настрою користувача.

```
// Приклад аналізу емоцій
async function analyzeEmotion(text) {
  const response = await axios.post('https://api.claude.ai/emotion-analysis', {
    text: text
  }, {
    headers: {
      'Authorization': 'Bearer YOUR_API_KEY'
    }
  });
  return response.data.emotion;
}
analyzeEmotion("I am so excited about this new feature!").then(emotion => {
  console.log(`Detected emotion: ${emotion}`);
});
```

Аналіз користувацької активності: Інтеграція Claude API для побудови моделей поведінки користувачів, що підвищує точність рекомендацій.

```
// Приклад аналізу активності користувачів
async function analyzeUserActivity(userData) {
  const response = await axios.post('https://api.claude.ai/user-activity', {
```

```
    data: userData
  }, {
    headers: {
      'Authorization': 'Bearer YOUR_API_KEY'
    }
  });
  return response.data.analysis;
}
analyzeUserActivity({ id: 123, actions: ["click", "scroll", "purchase"] }).then(analysis => {
  console.log(analysis);
});
```

Автоматизація рутинних завдань: Використання API для автоматичного створення звітів, аналізу даних або генерації контенту.

```
// Генерація автоматичного звіту
async function generateReport(data) {
  const response = await axios.post('https://api.claude.ai/report-generation', {
    input: data
  }, {
    headers: {
      'Authorization': 'Bearer YOUR_API_KEY'
    }
  });
  return response.data.report;
}
generateReport({ metrics: ["sales", "growth"], period: "last month" }).then(report => {
  console.log(report);
});
```

Ідентифікація об’єктів: Використання Claude API для обробки зображень, наприклад, у додатках для електронної комерції чи медицини.

```
// Приклад ідентифікації об’єктів
async function identifyObjects(imageUrl) {
  const response = await axios.post('https://api.claude.ai/object-detection', {
    image: imageUrl
  }, {
    headers: {
      'Authorization': 'Bearer YOUR_API_KEY'
    }
  });
  return response.data.objects;
}
identifyObjects("https://example.com/image.jpg").then(objects => {
  console.log(objects);
});
```

Переваги використання Claude API:

1. Швидка інтеграція. Простий синтаксис і документація дозволяють швидко реалізувати новий функціонал.
2. Гнучкість. Claude API адаптується до різних завдань, від обробки тексту до розпізнавання зображень.
3. Масштабованість. Можливість роботи з великими обсягами даних завдяки хмарній інфраструктурі.
4. Персоналізація. Забезпечує глибокий аналіз даних для створення персоналізованого досвіду.

Інтеграція Claude API відкриває нові горизонти для вдосконалення веб-додатків, роблячи їх більш інтелектуальними та зручними для користувачів..

4.3 Масштабування додатку та поліпшення бази даних Elasticsearch

Масштабування веб-додатку є ключовим аспектом його довготривалої стабільності та продуктивності. Це включає не лише збільшення можливостей серверної інфраструктури, але й адаптацію бази даних до зростаючих вимог. Elasticsearch, як високопродуктивна система управління даними, відіграє центральну роль у забезпеченні швидкого доступу до інформації, її аналізу та зберігання великих обсягів даних. Завдяки своїй модульній архітектурі та підтримці горизонтального і вертикального масштабування, Elasticsearch забезпечує оптимізацію обробки даних. Це дозволяє впроваджувати сучасні рішення, такі як інтелектуальний пошук, кластеризація документів і аналітика великих даних, які зберігають стабільність і продуктивність додатків навіть при стрімкому зростанні користувачів. У цьому підрозділі розглянуто підходи до масштабування додатку та оптимізації роботи Elasticsearch, включаючи практичні приклади й рекомендації для реалізації таких підходів.

Масштабування додатку

Горизонтальне масштабування: Додавання нових серверів для обробки запитів дозволяє забезпечити масштабованість і безперебійну роботу додатка. Використання балансувальників навантаження, таких як NGINX, HAProxy або AWS ELB, дає змогу ефективно розподіляти трафік між кількома серверами, запобігаючи перевантаженню окремих вузлів. Крім того, горизонтальне масштабування може бути розширене через інтеграцію з хмарними сервісами, які підтримують автоматичне масштабування залежно від поточного навантаження.

```
# Приклад конфігурації NGINX для балансування навантаження
upstream backend {
    server backend1.example.com;
    server backend2.example.com;
}
server {
    listen 80;
```

```
    location / {
        proxy_pass http://backend;
    }
}
```

Вертикальне масштабування: Збільшення ресурсів серверів (процесори, оперативна пам’ять) дозволяє значно підвищити продуктивність окремих вузлів. Хмарні сервіси, такі як AWS або Google Cloud, забезпечують швидке налаштування автоматичного масштабування під поточні потреби додатку. Наприклад, можна налаштувати збільшення ресурсів під час пікових навантажень:

```
# Приклад використання AWS CLI для зміни конфігурації EC2 інстансу
aws ec2 modify-instance-attribute \
    --instance-id i-1234567890abcdef0 \
    --instance-type "m5.large"
```

Цей підхід також включає інтеграцію з системами моніторингу, такими як CloudWatch, які автоматично тригерять масштабування залежно від метрик, таких як завантаження процесора або пам'яті.

Кластеризація: Використання кластерів є ключовим підходом для підвищення доступності, продуктивності та стійкості системи. Elasticsearch дозволяє розподіляти дані між вузлами кластера, що забезпечує безперервний доступ навіть у разі відмови окремих серверів. Завдяки цьому підходу можна обробляти великі обсяги запитів, автоматично балансувати навантаження між вузлами і масштабувати систему горизонтально. Elasticsearch також підтримує реплікацію даних, яка мінімізує ризик втрати інформації при збоях. Наприклад, можна налаштувати кілька вузлів у різних географічних регіонах для забезпечення високої доступності.

```
# Приклад конфігурації Elasticsearch кластера
```

```
cluster.name: my-cluster
node.name: node-1
network.host: 0.0.0.0
path.data: /var/lib/elasticsearch
path.logs: /var/log/elasticsearch
cluster.initial_master_nodes:
  - node-1
  - node-2
```

Поліпшення бази даних Elasticsearch

Оптимізація індексів: Використання правильних мапінгів і налаштувань індексів для скорочення витрат ресурсів і прискорення пошуку. Для цього варто застосовувати стиснення даних за допомогою параметра index.codec, наприклад zstd, який зменшує розмір індексів без значного впливу на продуктивність. Також рекомендується зменшувати кількість сегментів у шардових індексах через операції злиття (merge), що підвищує ефективність пошуку. Для складних полів варто застосовувати специфічні типи мапінгів, такі як keyword для агрегацій або dense_vector для векторних пошуків у машинному навчанні.

```
{
  "mappings": {
    "properties": {
      "name": { "type": "text" },
      "age": { "type": "integer" },
      "joined": { "type": "date" }
    }
  }
}
```

Впровадження життєвих циклів індексів: Автоматизація архівації старих даних або їх видалення для зменшення навантаження на систему є критично важливим кроком для підтримання ефективності Elasticsearch. Цей підхід дозволяє

зберігати актуальні дані в оперативному доступі, одночасно переміщуючи застарілі записи в архів чи видаляючи їх. Наприклад, за допомогою ILM (Index Lifecycle Management) можна автоматизувати переходи між фазами зберігання. Це включає гарячу фазу для активних даних, теплу фазу для рідше використовуваних записів, холодну фазу для архівів та фазу видалення. Така конфігурація мінімізує витрати на зберігання та покращує продуктивність пошуку.

```
{
  "policy": {
    "phases": {
      "hot": {
        "actions": {
          "rollover": {
            "max_size": "50gb",
            "max_age": "30d"
          }
        }
      },
      "delete": {
        "min_age": "90d",
        "actions": {
          "delete": {}
        }
      }
    }
  }
}
```

Кешування результатів пошуку: Збереження популярних запитів у кеші для прискорення їх виконання є важливим елементом оптимізації продуктивності Elasticsearch. Цей підхід дозволяє уникати повторного виконання ресурсоємних запитів, зберігаючи результати у пам’яті або на диску. Наприклад, використання

request_cache у Elasticsearch забезпечує автоматичне кешування результатів часто виконуваних пошуків:

```
# Активація кешування запитів для конкретного індексу
PUT /my-index/_settings
{
  "index.requests.cache.enable": true
}
```

Крім того, можна використовувати сторонні рішення, такі як Redis, для зовнішнього кешування результатів запитів:

```
// Збереження результатів у Redis
const redis = require('redis');
const client = redis.createClient();
async function cacheSearchResults(query, results) {
  const key = `search:${query}`;
  await client.set(key, JSON.stringify(results), 'EX', 3600); // Кеш на 1 годину
}
// Використання кешу
async function getCachedResults(query) {
  const key = `search:${query}`;
  const cachedResults = await client.get(key);
  return cachedResults ? JSON.parse(cachedResults) : null;
}
# Включення кешування в Elasticsearch
GET /_cache/clear
```

Моніторинг продуктивності: Використання Kibana та сторонніх інструментів, таких як Grafana, дозволяє в режимі реального часу відстежувати ключові метрики роботи Elasticsearch. Наприклад, можна налаштувати дашборди

для моніторингу використання CPU, пам'яті, затримок у виконанні запитів і пропускної здатності системи. Інтеграція з такими інструментами, як Prometheus, дає змогу зберігати історичні дані для аналізу та прогнозування можливих перевантажень. Нижче наведено приклад конфігурації експортеру метрик для Elasticsearch:

```
# Конфігурація Prometheus експортеру для Elasticsearch
es_exporter:
  metrics:
    - cluster_health
    - index_stats
    - node_stats
  scrape_interval: 10s
```

Такий підхід допомагає ідентифікувати вузькі місця та своєчасно оптимізувати роботу бази даних.

Масштабування та оптимізація Elasticsearch дозволяють ефективно обробляти великі обсяги даних, забезпечуючи стабільність, продуктивність і гнучкість веб-додатку. Це дає можливість впроваджувати сучасні рішення, такі як аналітика в реальному часі, швидкий повнотекстовий пошук та управління великими наборами даних, навіть у середовищах з інтенсивним навантаженням.

4.4 Висновки по розділу

У цьому розділі було розглянуто ключові шляхи удосконалення веб-додатку, зокрема масштабування інфраструктури, оптимізацію роботи бази даних Elasticsearch та інтеграцію нових можливостей Claude API. Основні висновки:

- Масштабування додатку: Горизонтальне та вертикальне масштабування, а також кластеризація дозволяють забезпечити стабільність і продуктивність додатку навіть за високих навантажень. Інтеграція хмарних технологій, автоматичне масштабування та

балансувальники навантаження значно полегшують управління ресурсами.

- Оптимізація Elasticsearch: Впровадження життєвих циклів індексів, кешування запитів, правильна настройка мапінгів та моніторинг продуктивності сприяють ефективному управлінню даними і підвищенню швидкодії системи.
- Інтеграція Claude API: Нові можливості, такі як обробка природної мови, аналітика користувацької активності та автоматизація рутинних завдань, розширюють функціонал додатку та покращують користувацький досвід.

Ці підходи та інструменти не лише покращують продуктивність і зручність додатку, але й забезпечують його довготривалу стабільність, масштабованість і відповідність сучасним вимогам ринку.

ВИСНОВКИ

У дипломній роботі було успішно виконано дослідження, спрямоване на досягнення поставленої мети: розробку теоретичної моделі та рекомендацій для створення веб-додатка пізнавальної гри на основі штучного інтелекту. У ході роботи було вирішено такі завдання:

Аналіз існуючих рішеньПроведено огляд сучасних технологій і платформ для розробки пізнавальних ігор, які використовують штучний інтелект. Виявлено основні виклики, серед яких інтеграція AI-алгоритмів, оптимізація продуктивності, забезпечення адаптивності до різних рівнів гравців і захист даних. Додатково розглянуто потенційні переваги використання генеративного AI для створення унікальних ігрових сценаріїв і динамічного контенту. Запропоновано алгоритмічні рішення для динамічної адаптації контенту, автоматичного створення завдань, інтеграції модулів навчання та підвищення інтерактивності геймплею. Проведений аналіз дозволив визначити найефективніші підходи до реалізації таких систем у веб-середовищі.

Рекомендації для реалізації рішеньРозроблено практичні рекомендації для створення веб-додатка пізнавальної гри, які охоплюють вибір технологій, інтеграцію алгоритмів штучного інтелекту, оптимізацію серверної інфраструктури та підвищення користувацького досвіду. Особливу увагу приділено створенню модульної архітектури, яка забезпечує легкість оновлення і масштабування. Впровадження алгоритмів персоналізації дозволяє адаптувати завдання до індивідуального рівня кожного користувача, що підвищує їх зацікавленість і мотивацію. Додатково розроблено рекомендації щодо інтеграції механізмів моніторингу продуктивності для своєчасного виявлення й усунення потенційних проблем. Запропоновані рекомендації спрямовані на забезпечення гнучкості, масштабованості, стабільності та безперебійної роботи додатка, а також на поліпшення користувацького досвіду через інтуїтивний інтерфейс і адаптивний дизайн.

Результати роботи підтвердили ефективність запропонованих підходів, зокрема у підвищенні адаптивності контенту, продуктивності системи та захищеності даних. Додатково розглянуто перспективи використання технологій штучного інтелекту для створення більш складних сценаріїв гри, що здатні адаптуватися до різних рівнів користувачів у режимі реального часу. Розроблені теоретичні моделі та рекомендації можуть бути інтегровані у реальні проєкти, включаючи освітні, тренінгові та розважальні програми, що доводить їхню практичну цінність та універсальність. Додаток демонструє високий потенціал для впровадження в системи дистанційного навчання, корпоративного тренінгу та інтерактивних платформ, відповідаючи сучасним вимогам до веб-рішень. Також запропоновані підходи сприяють зниженню витрат на розробку та забезпечують стабільність роботи системи навіть за умов зростаючого навантаження..

ПЕРЕЛІК ПОСИЛАНЬ

1. Murphy K. P. Machine Learning: A Probabilistic Perspective. – MIT Press, 2012. – 1067 p.
2. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. – Pearson, 2020. – 1136 p.
3. Bishop C. Pattern Recognition and Machine Learning. – Springer, 2006. – 738 p.
4. Kubernetes Documentation. Official Kubernetes Documentation. – [Online Resource], 2024.
5. Hightower K., Burns B., Beda J. Kubernetes: Up and Running. – O'Reilly Media, 2019. – 246 p.
6. Cerberus S. Kubernetes Patterns: Reusable Elements for Designing Cloud-Native Applications. – O'Reilly Media, 2020. – 320 p.
7. Docker Documentation. Official Docker Documentation. – [Online Resource], 2024.
8. Chen J., Guestrin C. XGBoost: A Scalable Tree Boosting System. – ACM, 2016. – 785–794 p.
9. Goldberg D. E. Genetic Algorithms in Search, Optimization, and Machine Learning. – Addison-Wesley, 1989. – 432 p.
10. Trivy Documentation. Open Source Security for Containers. – [Online Resource], 2024.
11. Prometheus Documentation. Monitoring and Alerting Toolkit. – [Online Resource], 2024.
12. Grafana Documentation. Open Source Analytics & Monitoring Solution. – [Online Resource], 2024.
13. Bottou L. Stochastic Gradient Descent Tricks. – Springer, 2012. – 421–436 p.
14. LeCun Y., Bengio Y., Hinton G. Deep Learning. – Nature, 2015. – 436–444 p.
15. Shannon C. E. A Mathematical Theory of Communication. – Bell System Technical Journal, 1948. – 379–423 p.

16. HashiCorp Vault Documentation. Secrets Management. – [Online Resource], 2024.
17. Gartner Report. Trends in Cloud-Native Application Security. – Gartner, 2023.
18. Dijkstra E. W. A Note on Two Problems in Connexion with Graphs. – Numerische Mathematik, 1959. – 269–271 p.

ДОДАТОК А. ПРЕЗЕНТАЦІЯ

1

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

ДИПЛОМНА РОБОТА
на ступінь вищої освіти магістр
із спеціальності 122 Комп'ютерні науки

ВЕБ-ДОДАТОК ПІЗНАВАЛЬНОЇ ГРИ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ

Виконав: студент 6 курсу, групи КНДМ-63
Проценко Ілля Юрійович
Керівник: викладач кафедри Комп'ютерних наук
д.ф., Кравчук П. О.

Київ - 2025

2

ЗАГАЛЬНА ХАРАКТЕРИСТИКА ДИПЛОМНОЇ РОБОТИ

Тема	Веб-додаток пізнавальної гри на основі штучного інтелекту
Мета дослідження	Розробка концептуальної моделі веб-додатку пізнавальної гри, яка базується на інтеграції штучного інтелекту для забезпечення інтерактивності, зручності та масштабованості
Наукове завдання	Розробка теоретичних основ і практичних рекомендацій щодо створення веб-додатків з інтерактивними та пізнавальними функціями на основі ШІ
Об'єкт дослідження	Процеси розробки веб-додатків з інтерактивними та пізнавальними функціями на основі ШІ
Предмет дослідження	Методи проєктування та оптимізації архітектури веб-додатку з використанням сучасних технологій

3

СУЧАСНІ ТРЕНДИ У РОЗРОБЦІ ПІЗНАВАЛЬНИХ ІГОР

У сучасному світі пізнавальні ігри набувають усе більшої популярності, оскільки вони поєднують освітні, розважальні та тренувальні елементи. Їхня мета полягає у розвитку ключових компетенцій, таких як логічне мислення, пам'ять, креативність, здатність вирішувати складні задачі та аналітичні навички. Технології ШІ здатні аналізувати тривалі тенденції в поведінці користувачів, що дозволяє розробникам краще зрозуміти потреби аудиторії та вдосконалювати сценарії гри. Завдяки цьому ігри стають не лише джерелом розваг, але й потужним освітнім інструментом, що формує глибший рівень залучення користувачів.

Технології для проєкту:

- **Angular** - цей фронтенд-фреймворк забезпечує створення динамічного, інтерактивного та адаптивного інтерфейсу користувача. .
- **NestJS** - серверний фреймворк на базі Node.js.
- **Socket.io** - бібліотека для роботи з WebSocket, що дозволяє реалізувати функціонал взаємодії в реальному часі.
- **Claude API** - потужний інструмент для інтеграції штучного інтелекту.
- **Elasticsearch** - база даних.

Дані технології були обрані через їх відповідність вимогам проєкту, здатність забезпечувати надійну роботу у високонавантажених умовах, адаптивність до змінних потреб користувачів і ефективність у реалізації складних інтерактивних сценаріїв. Вони дозволяють створювати продукти, які легко масштабуються, підтримують інтеграцію з різноманітними API та сервісами, і забезпечують швидке впровадження нових функцій. Їхнє поширене використання в сучасних розробках також гарантує доступність великої кількості ресурсів для навчання, активну підтримку спільнот, широкий вибір готових рішень та бібліотек, що значно спрощує процес впровадження. Крім того, ці технології пропонують інструменти для вдосконалення продуктивності, аналізу даних у реальному часі та забезпечення високого рівня безпеки, що критично важливо для сучасних інтерактивних додатків.

4

ВИКОРИСТАННЯ ELASTICSEARCH ДЛЯ РОБОТИ З ДАНИМИ

Elasticsearch є однією з найпоширеніших сучасних баз даних для роботи з великими обсягами інформації. Її популярність пояснюється здатністю забезпечувати надзвичайно швидкий доступ до даних завдяки високоефективній системі індексації, яка забезпечує обробку мільйонів записів за лічені секунди. Це критично важливо для інтерактивних додатків, пізнавальних ігор, аналітичних платформ і систем рекомендацій, де швидкість доступу до інформації є ключовим фактором. Elasticsearch забезпечує не тільки оперативність, але й надійність у виконанні пошукових запитів, гарантуючи високу точність результатів навіть у системах із величезними наборами даних. Її продуктивність дозволяє створювати додатки, які працюють плавно та ефективно, забезпечуючи користувачів найкращим досвідом.

Kibana (рис. 4.1) надає потужні інструменти для аналітики, які дозволяють створювати детальні дашборди, що відображають ключові тренди, статистичні дані та інші важливі метрики. Завдяки цьому розробники та адміністратори системи можуть відслідковувати продуктивність у реальному часі, оцінювати вплив змін у системі та приймати зважені стратегічні рішення. Крім того, Kibana дає можливість проводити глибокий аналіз поведінкових патернів користувачів, виявляти тенденції та оптимізувати ігровий процес на основі цих даних. У пізнавальних іграх, наприклад, Kibana може показувати детальний прогрес гравців, аналізувати, які завдання вони виконують найкраще, і виділяти області для покращення. Це підвищує рівень персоналізації та сприяє більш інтерактивній взаємодії між системою та користувачем, що в кінцевому підсумку збільшує залученість і ефективність гри

Рисунок 4.1 – Kibana

КОНЦЕПТУАЛЬНА МОДЕЛЬ ДОДАТКУ ТА ФУНКЦІОНАЛЬНІ ВИМОГИ

Інтерфейс користувача (UI)

Інтерактивний та адаптивний дизайн, що забезпечує зручність взаємодії з додатком. Ключові елементи інтерфейсу включають інформаційну панель, систему навігації, форми для введення даних, інтерактивні елементи гри, а також можливість персоналізації для кожного користувача. Реалізація персоналізації включає налаштування кольорової схеми, створення користувацьких профілів та інтеграцію мовних уподобань.

Серверна частина

Логіка додатку, яка відповідає за обробку даних, синхронізацію ігрового процесу між користувачами, захист від несанкціонованого доступу та обробку запитів у реальному часі. Вона включає обробку асинхронних запитів, оптимізацію кешування для зменшення затримок, а також забезпечення розподіленого управління ресурсами для стабільної роботи за високих навантажень. .

База даних

Elasticsearch, яка забезпечує зберігання, індексацію та швидкий доступ до інформації, такої як профілі користувачів, результати ігор, метадані та лог-файли. Її потужності дозволяють виконувати складні пошукові запити та аналіз даних у реальному часі, підтримуючи динамічну взаємодію між компонентами додатку..

Інтеграція API

Claude API використовується для реалізації потужних алгоритмів штучного інтелекту, які адаптують ігровий процес до індивідуальних потреб кожного користувача, забезпечуючи персоналізацію і високий рівень залучення.

ФУНКЦІОНАЛЬНІ ВИМОГИ

Реалізація веб-додатку базується на таких функціональних вимогах, які забезпечують його ефективність та відповідність сучасним стандартам програмного забезпечення:

Реєстрація та авторизація користувачів

Цей модуль дозволяє створювати нові облікові записи, входити в систему та керувати профілями користувачів. Застосування сучасних протоколів безпеки, таких як OAuth або JWT, гарантує захист особистих даних, шифрування паролів і запобігання несанкціонованому доступу. Крім того, модуль передбачає функції відновлення доступу, багатофакторної автентифікації та перевірки активності сеансів, що забезпечує додатковий рівень безпеки.

Ігровий процес

Основний акцент зроблено на інтерактивних завданнях, які адаптуються до рівня знань і досвіду кожного користувача. Завдяки використанню алгоритмів штучного інтелекту, система може автоматично коригувати складність завдань, пропонуючи оптимальний рівень виклику для кожного гравця. Кожне завдання доповнюється контекстуальною підтримкою, яка допомагає користувачам зрозуміти правила і цілі гри. Крім того, механізми гейміфікації, такі як нарахування балів, створення досягнень та інтерактивні підказки, сприяють підвищенню залученості та мотивації.

Синхронізація даних у реальному часі

Всі ігрові події та дії користувачів оновлюються миттєво завдяки інтеграції з Socket.io. Це забезпечує безперебійну взаємодію між учасниками гри, створюючи динамічне середовище для змагань та співпраці.

Аналітика та статистика

Додаток надає розширені інструменти для візуалізації прогресу користувачів у вигляді графіків, таблиць та інтерактивних дашбордів. Зібрані дані дозволяють створювати детальні аналітичні звіти, які допомагають користувачам оцінювати свій розвиток, визначати сильні сторони та області для покращення.

Масштабованість

Архітектура додатку розроблена таким чином, щоб забезпечити стабільну роботу навіть при різкому збільшенні кількості користувачів.

АСПЕКТИ КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ

Архітектура системи пізнавальної гри побудована за принципом клієнт-серверної взаємодії, яка забезпечує чіткий розподіл функцій між клієнтською частиною та сервером.

Модульність

Чіткий поділ на клієнтську і серверну частини спрощує розробку та підтримку системи, дозволяючи кожній частині виконувати специфічні завдання незалежно одна від одної. Такий підхід сприяє легкості масштабування, тестування та оновлення функціоналу без значних змін у загальній структурі додатку.

Реактивність

Використання WebSocket для реальної взаємодії між клієнтами та сервером дозволяє оновлювати дані миттєво та забезпечувати постійний зв'язок у режимі реального часу. Це дозволяє системі реагувати на дії користувачів з мінімальною затримкою, синхронізувати ігрові події для всіх учасників та зменшити навантаження на сервер завдяки оптимізації обміну даними. Реактивність також сприяє покращенню користувацького досвіду за рахунок плавної роботи навіть під час пікових навантажень.

Безпека

Захищені канали зв'язку між клієнтом і сервером (HTTPS, WebSocket Secure) забезпечують шифрування переданої інформації, що мінімізує ризик перехоплення даних або їх зміни під час передачі. Додатково реалізовані механізми багатофакторної автентифікації, перевірка цілісності даних та регулярний моніторинг потенційних загроз. Система підтримує сучасні криптографічні алгоритми, а також активне виявлення та запобігання атакам, включаючи DDoS та MITM. Це гарантує максимальний рівень безпеки для користувачів та стабільність роботи додатку.

Архітектура клієнт-серверної взаємодії забезпечує стабільність роботи додатку завдяки ефективному управлінню ресурсами, оптимізації потоків даних та розподілу навантаження між клієнтською і серверною частинами. Вона дозволяє легко масштабувати функціональність системи шляхом додавання нових модулів чи інтеграції з сучасними сервісами, такими як хмарні обчислення, інструменти машинного навчання або сервіси для аналізу великих даних. .

ІНТЕГРАЦІЯ ШТУЧНОГО ІНТЕЛЕКТУ ЧЕРЕЗ CLAUDE API

Інтеграція штучного інтелекту (ШІ) у веб-додаток через Claude API є ключовим елементом, що значно підвищує функціональність ігрової системи. Claude API надає потужні інструменти для реалізації адаптивного ігрового досвіду, аналізу поведінки користувачів, автоматизації складних процесів і впровадження інтерактивних інновацій. Це забезпечує створення інтуїтивно зрозумілого продукту, здатного не лише адаптуватися до потреб користувачів, але й передбачати їхні очікування за допомогою прогнозувальних моделей. Застосування Claude API дає змогу створювати інтерактивне середовище, яке в реальному часі аналізує багатофакторні дані, такі як ігрові метрики, активність, індивідуальні уподобання користувачів, і на основі цього генерує оптимізовані сценарії гри. Це сприяє формуванню не лише високого рівня залучення, але й постійного розвитку користувацького досвіду, підвищуючи задоволення від гри та стимулюючи глибоку емоційну прив'язаність до платформи. Крім того, система підтримує багаторівневу адаптацію контенту, яка дозволяє враховувати зміни в поведінці гравців та динамічно налаштовувати ігрові механіки відповідно до змінних умов.

```
// Приклад інтеграції Claude API у Node.js за допомогою axios
const axios = require('axios');
const API_KEY = 'your_claude_api_key';
const BASE_URL = 'https://api.claude.ai';
// Функція для надсилання запиту до Claude API
async function analyzeUserData(userData) {
    try {
        const response = await axios.post(`${BASE_URL}/analyze`, {
            data: userData,
        }, {
            headers: {
                'Authorization': `Bearer ${API_KEY}`,
                'Content-Type': 'application/json',
            },
        });
        return response.data; // Повернення результату аналізу
    } catch (error) {
        console.error('Помилка інтеграції з Claude API:', error);
        throw error;
    }
}
// Використання функції для аналізу даних користувача
const userData = {
    metrics: {
        activity: 85,
        successRate: 78,
        engagement: 90,
    },
    preferences: ['adventure', 'strategy'],
};
analyzeUserData(userData)
    .then(result => {
        console.log('Результати аналізу:', result);
    })
    .catch(error => {
        console.error('Помилка під час аналізу даних:', error);
    });
```

ІНТЕГРАЦІЯ ШТУЧНОГО ІНТЕЛЕКТУ ЧЕРЕЗ CLAUDE API 9

Claude API використовується для аналізу даних, зібраних під час взаємодії користувачів із системою. Цей аналіз охоплює різноманітні аспекти, включаючи частоту дій гравців, час виконання завдань, рівень взаємодії з ключовими елементами гри, а також їхні успіхи чи труднощі на різних етапах. Система використовує ці дані для формування багаторівневих аналітичних моделей, які дозволяють прогнозувати потреби гравців з високою точністю. На основі отриманих висновків система динамічно адаптує рівень складності завдань, створює персоналізовані ігрові сценарії, генерує релевантні рекомендації для покращення досвіду користувача та оптимізує порядок подання ігрового контенту. Завдяки цьому підходу досягається не лише високий рівень залучення, але й створюється довготривалий інтерес до гри, формуючи стійку емоційну прив'язаність гравців до платформи та стимулюючи їх повернення

```
# Приклад інтеграції Claude API у Python
import requests
API_KEY = 'your_claude_api_key'
BASE_URL = 'https://api.claude.ai'
def analyze_user_data(user_data):
    try:
        headers = {
            'Authorization':           f'Bearer
{API_KEY}',
            'Content-Type':
'application/json',
        }
        response                              =
requests.post(f'{BASE_URL}/analyze',   json={'data':
user_data}, headers=headers)
        response.raise_for_status()
        return response.json()
    except
requests.exceptions.RequestException as e:
        print(f'Error  integrating  with  Claude
API: {e}')
        return None
# Приклад даних користувача
user_data = {
    'metrics': {
        'activity': 85,
        'successRate': 78,
        'engagement': 90,
    },
    'preferences': ['adventure', 'strategy'],
}
result = analyze_user_data(user_data)
if result:
    print('Результати аналізу:', result)
else:
    print('Не   вдалося   отримати   результати
аналізу.')
```

ОПТИМІЗАЦІЯ ШВИДКОДІЇ КЛІЄНТСЬКОЇ І СЕРВЕРНОЇ ЧАСТИН 10

Оптимізація швидкодії є важливим етапом у розвитку веб-додатків, оскільки забезпечує покращення досвіду користувачів і підвищення продуктивності системи. Цей процес включає як технічні аспекти, такі як оптимізація коду, архітектури та баз даних, так і використання сучасних інструментів для моніторингу та аналізу. Основними завданнями оптимізації є зменшення затримок, прискорення обробки даних і забезпечення стабільності навіть за умов високих навантажень. У цьому підрозділі детально розглянемо ключові підходи до вдосконалення клієнтської та серверної частин, інтеграцію передових практик програмування, впровадження механізмів кешування та використання асинхронних технологій для обробки великих обсягів даних у реальному часі. Розглянемо також оптимізацію інфраструктури, включаючи масштабування серверів і інтеграцію з хмарними платформами.

Зменшення розміру ресурсів: Використання методів, таких як мініфікація JavaScript, CSS та HTML, а також стиснення зображень за допомогою сучасних форматів (WebP, AVIF), значно зменшує час завантаження сторінок. Наприклад, можна використовувати спеціалізовані інструменти, такі як Terser для JavaScript, PostCSS для CSS або ImageMagick для оптимізації зображень. Впровадження таких підходів дозволяє знизити об'єм переданих даних та зменшити час відгуку сервера, що позитивно впливає на швидкодію додатку.

Впровадження механізмів кешування: Браузерне кешування дозволяє повторно використовувати завантажені раніше ресурси, скорочуючи час завантаження нових сторінок. Для серверного кешування можна використовувати такі підходи, як кешування результатів обчислень або запитів до бази даних у Redis або Memcached. Це значно зменшує навантаження на сервер і прискорює час відповіді.

Lazy Loading: Завантаження лише тих ресурсів, які потрібні для поточного відображення, значно підвищує швидкодію додатку. Цей підхід дозволяє зменшити початковий час завантаження сторінки та зменшити споживання ресурсів. Lazy Loading активно використовується в фреймворках, таких як Angular та React. У Angular, наприклад, модулі можуть завантажуватися динамічно

```
// Приклад мінімфікації JavaScript за допомогою
Terser
const Terser = require('terser');
const    jsCode    =    `function    helloWorld()    {
console.log("Hello World"); }`;
const minifiedCode = Terser.minify(jsCode).code;
console.log(minifiedCode);
// Приклад реалізації кешування запиту до бази
даних
const getCachedData = async (key, queryFunc) => {
  const cached = await redisClient.get(key);
  if (cached) {
    return JSON.parse(cached);
  }
  const data = await queryFunc();
  redisClient.set(key,  JSON.stringify(data),  'EX',
3600);
  return data;
};
const fetchData = async () => {
  return  await  getCachedData('user:123',  async  ()
=> {
    return db.query('SELECT * FROM users WHERE id =
123');
  });
};
// HTTP-заголовок для кешування
response.setHeader('Cache-Control', 'public, max-
age=31536000');

 // Angular: Приклад Lazy Loading для маршруту
 const routes: Routes = [
   {   path:   'dashboard',   loadChildren:   ()   =>
 import('./dashboard/dashboard.module').then(m           =>
 m.DashboardModule) },
 ];
 @NgModule({
   imports: [RouterModule.forRoot(routes)],
   exports: [RouterModule]
 })
 export class AppRoutingModule {}
```

РОЗШИРЕННЯ ФУНКЦІОНАЛУ НА ОСНОВІ НОВИХ МОЖЛИВОСТЕЙ CLAUDE API

11

Claude API пропонує широкий спектр можливостей для інтеграції штучного інтелекту у веб-додатки, дозволяючи створювати інноваційний функціонал, який покращує користувацький досвід та розширює потенціал системи. У цьому підрозділі розглянемо основні підходи до розширення функціоналу за допомогою Claude API та практичні приклади його застосування.

Основні можливості Claude API:

• Обробка природної мови (NLP): Claude API дозволяє створювати інтелектуальних чат-ботів, які розуміють контекст користувацьких запитів і можуть генерувати відповідні відповіді.

• Аналіз даних: Інструменти для класифікації, оцінювання тональності тексту та пошуку ключових фраз у великих обсягах даних.

• Машинний переклад: Реалізація багатомовної підтримки для інтерфейсу додатку.

• Побудова рекомендаційних систем: Використання моделей Claude для аналізу поведінки користувачів і надання персоналізованих рекомендацій.

Розпізнавання образів: Аналіз зображень для автоматизації процесів, наприклад, ідентифікація об'єктів чи розпізнавання тексту

Розширення функціоналу

Інтелектуальні відповіді: Додаток може використовувати Claude API для надання детальних і контекстно залежних відповідей у чатах.

Розпізнавання емоцій: Використання API для аналізу тональності тексту дозволяє адаптувати інтерфейс додатку до настрою користувача.

Автоматизація рутинних завдань: Використання API для автоматичного створення звітів, аналізу даних або генерації контенту.

```
// Приклад інтелектуальних відповідей
async function getSmartReply(inputMessage) {
  const response = await axios.post('https://api.claude.ai/smart-reply', {
    message: inputMessage
  }, {
    headers: {
      'Authorization': 'Bearer YOUR_API_KEY'
    }
  });
  return response.data.reply;
}
getSmartReply("How can I reset my password?").then(reply => {
  console.log(reply);
});
```

```
// Інтеграція Claude API для обробки тексту
const axios = require('axios');
async function analyzeText(text) {
  const response = await axios.post('https://api.claude.ai/analyze', {
    text: text
  }, {
    headers: {
      'Authorization': 'Bearer YOUR_API_KEY'
    }
  });
  return response.data;
}
analyzeText("What is the weather like today?").then(result => {
  console.log(result);
});
// Використання Claude API для генерації тексту
async function generateText(prompt) {
  const response = await axios.post('https://api.claude.ai/generate', {
    prompt: prompt
  }, {
    headers: {
      'Authorization': 'Bearer YOUR_API_KEY'
    }
  });
  return response.data.text;
}
generateText("Напиши огляд про застосунок штучного інтелекту.").then(text => {
  console.log(text);
});
```

МАСШТАБУВАННЯ ДОДАТКУ ТА ПОЛІПШЕННЯ БАЗИ ДАНИХ ELASTICSEARCH

12

Масштабування веб-додатку є ключовим аспектом його довготривалої стабільності та продуктивності. Це включає не лише збільшення можливостей серверної інфраструктури, але й адаптацію бази даних до зростаючих вимог. Elasticsearch, як високопродуктивна система управління даними, відіграє центральну роль у забезпеченні швидкого доступу до інформації, її аналізу та зберігання великих обсягів даних. Завдяки своїй модульній архітектурі та підтримці горизонтального і вертикального масштабування, Elasticsearch забезпечує оптимізацію обробки даних. Це дозволяє впроваджувати сучасні рішення, такі як інтелектуальний пошук, кластеризація документів і аналітика великих даних, які зберігають стабільність і продуктивність додатків навіть при стрімкому зростанні користувачів. У цьому підрозділі розглянуто підходи до масштабування додатку та оптимізації роботи Elasticsearch, включаючи практичні приклади й рекомендації для реалізації таких підходів.

Масштабування додатку

Горизонтальне масштабування: Додавання нових серверів для обробки запитів дозволяє забезпечити масштабованість і безперебійну роботу додатка. Використання балансувальників навантаження, таких як NGINX, HAProxy або AWS ELB, дає змогу ефективно розподіляти трафік між кількома серверами, запобігаючи перевантаженню окремих вузлів. Крім того, горизонтальне масштабування може бути розширене через інтеграцію з хмарними сервісами, які підтримують автоматичне масштабування залежно від поточного навантаження.

Поліпшення бази даних Elasticsearch

Оптимізація індексів: Використання правильних мапінгів і налаштувань індексів для скорочення витрат ресурсів і прискорення пошуку. Для цього варто застосовувати стиснення даних за допомогою параметра index.codec, наприклад zstd, який зменшує розмір індексів без значного впливу на продуктивність. Також рекомендується зменшувати кількість сегментів у шардових індексах через операції злиття (merge), що підвищує ефективність пошуку. Для складних полів варто застосовувати специфічні типи мапінгів, такі як keyword для агрегацій або dense_vector для векторних пошуків у машинному навчанні.

```
# Приклад конфігурації NGINX для балансування навантаження
upstream backend {
    server backend1.example.com;
    server backend2.example.com;
}
server {
    listen 80;

    location / {
        proxy_pass http://backend;
    }
}
# Приклад використання AWS CLI для зміни конфігурації EC2 інстансу
aws ec2 modify-instance-attribute \
    --instance-id i-1234567890abcdef0 \
    --instance-type "m5.large«
{
  "mappings": {
    "properties": {
      "name": { "type": "text" },
      "age": { "type": "integer" },
      "joined": { "type": "date" }
    }
  }
}
// Збереження результатів у Redis
const redis = require('redis');
const client = redis.createClient();
async function cacheSearchResults(query, results) {
  const key = `search:${query}`;
  await client.set(key, JSON.stringify(results), 'EX', 3600); // Кеш на 1 годину
}
// Використання кешу
async function getCachedResults(query) {
  const key = `search:${query}`;
  const cachedResults = await client.get(key);
  return cachedResults ? JSON.parse(cachedResults) : null;
}
# Включення кешування в Elasticsearch
GET /_cache/clear
```

ВИСНОВКИ

Було успішно виконано дослідження, спрямоване на досягнення поставленої мети:

1. Проведено аналіз сучасних технологій і рішень для створення пізнавальних ігор на основі штучного інтелекту. Виявлено ключові виклики, такі як складність інтеграції AI-алгоритмів, високі вимоги до обчислювальних ресурсів, забезпечення адаптивності до різних рівнів користувачів і гарантування стабільної роботи веб-додатку. Запропоновано алгоритмічні підходи для адаптації контенту, підвищення продуктивності та інтерактивності ігрового процесу.

2. Розроблено багаторівневу модель безпеки для веб-додатку, яка включає аутентифікацію користувачів, захист персональних даних, шифрування передачі даних між сервером і клієнтом, а також виявлення потенційних загроз. Проведений аналіз показав, що впровадження цієї моделі мінімізує ризики несанкціонованого доступу та підвищує надійність збереження інформації.

3. Розроблено рекомендації для створення та впровадження пізнавальних ігор з використанням штучного інтелекту, які охоплюють вибір моделей AI, оптимізацію роботи серверної частини та забезпечення плавності взаємодії користувачів із системою. Запропоновані рішення спрямовані на підвищення зацікавленості користувачів, ефективного використання ресурсів і забезпечення безперебійної роботи додатку.