ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку для генерації адаптивних сюжетних ліній з використанням штучного інтелекту»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Микола МАЦЕЛА
(підпис)

Виконав: здобувач вищої освіти групи ПД-43
Микола МАЦЕЛА

Керівник: Юрій ЗАДОНЦЕВ
канд.техн. наук

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Ірина ЗАМРІЙ
« ____» _____2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Мацелі Миколі Романовичу

1. Тема кваліфікаційної роботи: «Розробка web-застосунку для генерації адаптивних сюжетних ліній з використання штучного інтелекту»

керівник кваліфікаційної роботи канд. техн. наук Юрій ЗАДОНЦЕВ,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 р. № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про принципи роботи великих мовних моделей (LLM) та методи їхнього налаштування (prompt engineering), основи процедурної генерації контенту в ігрових системах, архітектурні патерни побудови хмарних застосунків, технічна документація з опису фреймворку Next.js, середовища Supabase, API-інтерфейсів для інференсу нейронних мереж (Groq Cloud) та принципів безпеки при роботі з генеративним контентом.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Провести аналіз предметної галузі для дослідження методів генерації динамічних сюжетів та процедурної генерації контенту та інтерактивних наративів.
2. Провести порівняльний аналіз великих мовних моделей Llama 3.3, GPT-4 та

Claude для вибору оптимального двигуна.
3. Провести аналіз існуючих методів, технологій та інструментів для реалізації інтелектуальної системи генерації динамічних сюжетів.
4. Провести проєктування архітектури, розробити схеми взаємодії між Frontend, Edge-функціями та AI-провайдером.
5. Реалізувати інтелектуальну систему генерації динамічних сюжетів.
6. Провести тестування кінцевого вигляду продукту та перевірку працездатності реалізованого функціоналу.

1. Перелік графічного матеріалу: *презентація*
    1. Аналіз аналогів.
    2. Вимоги до функціональності та технічні обмеження системи.
    3. Програмні засоби реалізації.
    4. Діаграма прецедентів.
    5. Діаграма компонентів.
    6. Діаграма послідовності процесу обробки ігрового ходу.
    7. Діаграма діяльності.
    8. Екранні форми.
    9. Апробація результатів дослідження.

2. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури з методів генерації та LLM	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів генеративних ігрових систем	01.03-07.03.2026	
3	Огляд та аналіз технологій інтеграції мовних моделей у веб-застосунки	08.03-17.03.2026	
4	Проєктування архітектури системи для динамічної генерації ігрового контексту	18.03-27.03.2026	
5	Програмна реалізація та документування веб-застосунку з інтеграцією LLM та хмарних баз даних	28.03-10.04.2026	
6	Тестування програмного забезпечення на основі типових сценаріїв користувача	11.04-21.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	22.04-29.04.2026	
8	Розробка демонстраційних матеріалів	30.04-10.05.2026	
9	Попередній захист роботи	11.05-22.05.2026	

Здобувач вищої освіти _____ Микола МАЦЕЛА
(підпис)

Керівник

кваліфікаційної роботи _____ Юрій ЗАДОНЦЕВ
(підпис

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 56 стор., 1 табл., 3 рис., 26 джерел.

Мета роботи: вдосконалення процесу ігрової взаємодії шляхом впровадження інтелектуальної системи генерації динамічних сюжетів.

Об'єкт дослідження: процес генерації та управління динамічними сюжетними лініями в цифрових ігрових системах.

Предмет дослідження: методи, технології та інструменти створення інтелектуальної системи генерації динамічних сюжетів.

Короткий зміст роботи: у роботі проаналізовано методи процедурної генерації контенту для створення адаптивних ігрових наративів. Розроблено архітектуру веб-застосунку та реалізовано ключові функції: динамічне керування ігровим контекстом, інтеграцію з LLM через API та систему автентифікації користувачів. Проведено функціональне тестування системи. У проєкті використано фреймворк Next.js, середовище Supabase для зберігання даних та протоколи потокової передачі для мінімізації затримки при генерації сюжету.

Сферою використання застосунку є автоматизація процесу створення інтерактивних рольових ігор, що забезпечує користувачам можливість взаємодії з гнучким та адаптивним штучним інтелектом-майстром без потреби у попередній підготовці складних сценарних структур.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, ПРОЦЕДУРНА ГЕНЕРАЦІЯ, НАРАТИВ, ВЕЛИКІ МОВНІ МОДЕЛІ (LLM), ШТУЧНИЙ ІНТЕЛЕКТ, ГЕЙМ-МАЙСТЕР, NEXT.JS, REACT, SUPABASE, POSTGRESQL, JSONB, GROQ API, АВТЕНТИФІКАЦІЯ, РЕАКТИВНА АРХІТЕКТУ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 9

ВСТУП 10

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ НАРАТИВІВ 12

1.1 Алгоритмічні обмеження детермінованого моделювання сценаріїв у програмній інженерії 12

1.2 Вплив обмежень детермінованих алгоритмів на користувацький досвід та життєвий цикл розробки 13

1.3 Існуючі технологічні рішення 15

1.4 Аналіз аналогів 16

1.5 Роль генеративних систем у трансформації цифрового ігрового простору 18

1.6 Представлення ігрового контексту в цифровій формі 18

1.7 Дослідження проблеми семантичної ентропії та деградації контексту в довготривалих ігрових сесіях 26

2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ ТА КОМПОНЕНТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 29

2.1 Обґрунтування функціональних та нефункціональних вимог до системи 30

2.2 Моделювання прецедентів взаємодії користувача з системою 35

2.3 Архітектурна модель та взаємодія компонентів системи 36

2.4 Проектування логічної структури бази даних 37

2.5 Проектування інтерфейсу користувача 38

2.6. Інженерія системного промпту в архітектурі ШІ-Майстра 40

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ 43

3.3 Тестування програмного забезпечення та аналіз метрик продуктивності 45

3.4 Програмна реалізація механізмів потокової передачі даних (Streaming) та

ЗМІСТ

парсингу контенту 47

3.5 Реалізація системи автентифікації користувачів та розмежування доступу на рівні даних 48

3.6 Оптимізація клієнтського рендерингу та управління скролом чату 49

4 РОЗГОРТАННЯ, ЕКСПЛУАТАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 50

4.1.Аналіз вимог до апаратного забезпечення та середовища виконання 50

4.2. Архітектура хмарного розгортання та інфраструктурні рішення 51

4.3. Налаштування бази даних та інтеграція з LLM 51

4.4. Керівництво користувача та сценарії взаємодії 52

4.5. Заходи щодо забезпечення інформаційної безпеки та відмовостійкості 53

4.6. Економічна оцінка проекту та перспективи монетизації 54

ВИСНОВКИ 55

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 57

ДОДАТОК А. ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ 59

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ 67

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

LLM (Large Language Model) – велика мовна модель.

API (Application Programming Interface) – інтерфейс програмування застосунків.

REST (Representational State Transfer) – архітектурний стиль передачі стану представлення.

JSON (JavaScript Object Notation) – текстовий формат обміну даними.

JWT (JSON Web Token) – стандарт створення токенів доступу.

SQL (Structured Query Language) – мова структурованих запитів.

LPU (Language Processing Unit) – спеціалізований процесор для прискорення інференсу мовних моделей.

RLS (Row Level Security) – механізм безпеки на рівні рядків у базі даних.

CI/CD (Continuous Integration / Continuous Deployment) – безперервна інтеграція та безперервна доставка.

URL (Uniform Resource Locator) – уніфікований локатор ресурсів.

HTTP (Hypertext Transfer Protocol) – протокол передачі гіпертексту.

DTO (Data Transfer Object) – об’єкт передачі даних.

SPA (Single Page Application) – односторінковий веб-застосунок.

СУБД – система управління базами даних.

ВСТУП

Актуальність: Сфера розробки інтерактивного програмного забезпечення та систем цифрових розваг перебуває на етапі переходу від жорстко детермінованих архітектур до механізмів процедурної генерації. Традиційні сюжетно-орієнтовані застосунки функціонують на базі попередньо визначених графів станів та сценарних дерев. Такий інженерний підхід вимагає значних ресурсів на етапі розробки та критично обмежує варіативність взаємодії, оскільки кінцевий користувач змушений рухатися виключно закладеними програмістами маршрутами. Інтеграція великих мовних моделей (LLM) дозволяє перевести процес формування наративу в режим реального часу. Генеративні системи здатні аналізувати нестандартний користувацький ввід і динамічно адаптувати стан віртуального світу. Однак практичне впровадження таких рішень у веб-середовище супроводжується інфраструктурними проблемами: високою затримкою генерації (latency), яка нівелює ефект занурення, та обмеженістю контекстного вікна, що призводить до руйнування логіки сюжету при тривалих сесіях. Вирішення цих проблем шляхом використання апаратних прискорювачів інференсу (Groq LPU) та ефективних механізмів управління станом бази даних (JSONB у PostgreSQL) є своєчасним та актуальним науково-практичним завданням.

Об'єктом дослідження є процес генерації та управління нелінійними сюжетними лініями в цифрових ігрових системах на основі користувацького вводу. *Предметом дослідження* є методи та технології інтеграції великих мовних моделей (LLM) у веб-застосунки для забезпечення логіки автономного управління ігровим наративом.

Метою роботи є розробка інтерактивного веб-застосунку з використанням методів штучного інтелекту для створення адаптивних нелінійних наративів у реальному часі.

Методи дослідження: У процесі виконання роботи використано комплекс методів наукового пізнання та інженерного проектування: системний аналіз (при

формуванні вимог до програмного забезпечення та порівнянні аналогів); об’єктно-орієнтований підхід (при розробці архітектури клієнтської частини на базі React); методи реляційного моделювання (для проектування схем бази даних PostgreSQL); методи емпіричного тестування (для перевірки метрик затримки та відмовостійкості системи).

Далі було створено функціонуючий прототип веб-застосунку «StoryTaller», який забезпечує генерацію сюжетного контенту з мінімальною затримкою. Застосування формату зберігання даних JSONB у середовищі Supabase дозволило вирішити проблему швидкого відновлення багатовимірного ігрового контексту. Розроблені архітектурні рішення можуть бути використані як самостійний продукт або імплементовані як модуль ШІ-генерації у складніші ігрові рушії

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ НАРАТИВІВ

Створення нелінійного сюжету залишається однією з найскладніших задач під час розробки інтерактивних продуктів. Гравці вимагають персоналізованого контенту, але часто отримують лінійні та передбачувані сценарії.

Розробники витрачають багато часу на написання розгалужених діалогів, проте дії користувача рідко мають реальний вплив на стан віртуального світу. Виникає явище «ілюзії вибору». З появою продуктивних генеративних нейромереж з'явилася можливість створювати системи, де текст не завантажується з готової бази даних, а генерується динамічно під час сесії. Цей розділ присвячено дослідженню інженерних недоліків класичних детермінованих моделей та аналізу рішень на базі великих мовних моделей (LLM). На основі цього буде сформовано архітектурні вимоги до нової наративної системи.

1.1 Алгоритмічні обмеження детермінованого моделювання сценаріїв у програмній інженерії

У більшості ігрових рушіїв логіка квестів та діалогів будується на детермінованих алгоритмах. Найчастіше для цього використовують скінченні автомати (Finite State Machines, FSM) або графи станів. З точки зору архітектури ми маємо набір вузлів (V) та ребер (E). Кожен вузол відповідає за фіксований стан системи — конкретну репліку NPC або локацію. Переходи між ними відбуваються строго по ребрах графа через визначені тригери або прямий вибір в інтерфейсі.

Проблема такого підходу очевидна — комбінаторний вибух. Коли інженер намагається дати гравцю більше свободи на початку сесії, кількість можливих станів системи починає зростати експоненційно. Підтримувати цілісність такого масиву даних дуже складно.

Доводиться вручну прописувати тисячі зв'язків і тестувати кожну гілку, щоб уникнути логічних помилок. Наприклад, щоб NPC не згадував подію, якої гравець ще не бачив у своїй гілці проходження. Через великі витрати на розробку більшість студій використовують «наративні рейки». Вони звужують сотні дрібних рішень користувача до кількох наперед заданих фіналів, суттєво обмежуючи реальну варіативність.

Вирішенням проблеми масштабованості є процедурна генерація контенту (Procedural Content Generation) у реальному часі. Тут замість статичних вузлів використовуються генеративні моделі та логічні правила. Базовим рушієм для цього зараз виступають великі мовні моделі (LLM).

Але інтеграція LLM у клієнтські застосунки, зокрема в архітектуру Single Page Applications (SPA), створює нові проблеми для розробки бекенду та управління станом:

1. Ефемерність стану. LLM працюють через API за принципом «запит-відповідь» і є stateless-системами. Модель не має пам'яті між тактами обчислень. Якщо їй не передати історію взаємодії заново, вона втрачає контекст попередніх кроків.

2. Ліміт токенів (Context Window). Кожна модель має жорстке апаратне обмеження на обсяг вхідних даних для одного запиту.

3. Деградація контексту. Історія сесії постійно зростає. Якщо відправляти її монолітно, ліміт токенів швидко вичерпається. Крім того, це збільшує час обробки на сервері та підвищує вартість запитів. З часом базові системні інструкції (наприклад, правила світу гри) просто витісняються з пам'яті моделі.

1.2 Вплив обмежень детермінованих алгоритмів на користувацький досвід та життєвий цикл розробки

Обмеження жорстко прописаних сценаріїв прямо погіршують продуктові метрики. Користувачі часто стикаються з неможливістю виконати очевидну дію просто тому, що розробники не додали цей варіант у граф станів. Гравцю

доводиться обирати серед нерелевантних пропозицій інтерфейсу. Це миттєво руйнує ефект занурення (immersion) і робить поведінку системи штучною.

З інженерної точки зору складно забезпечити реграбельність (replayability). Статичні тексти і фіксовані тригери призводять до того, що за кілька ігрових сесій користувач повністю вивчає всі маршрути. Інтерес швидко зникає. Як наслідок, обвалюються ключові метрики — утримання аудиторії (Retention Rate) та тривалість життєвого циклу користувача (LTV). Подальша підтримка і генерування нового контенту для такого продукту стають економічно невигідними.

Відсутність динамічної адаптації безпосередньо знижує інтерес гравця. Користувачі витрачають час на читання однотипних діалогів. Вони не можуть застосувати власні креативні підходи до вирішення ігрових завдань. Аналіз ринку показує загальну нездатність детермінованих систем підлаштовуватися під стиль проходження чи настрій конкретного гравця.

На рівні життєвого циклу розробки (Software Development Life Cycle, SDLC) виникає проблема комбінаторного вибуху. Спроба описати всі можливі гілки дій геометрично збільшує навантаження на команду. Логіка вихідного коду ускладнюється. Виникають програмні тупики (deadlocks) або зациклення в графі станів. Відділ QA змушений витрачати тижні на ручне тестування кожної гілки. Перевірка сумісності станів вимагає написання тисяч тест-кейсів. Це різко піднімає вартість фінальної розробки та підтримки коду.

Відсутність інструментів генерації в реальному часі (Real-time Generation) приносить суб'єктивні труднощі для гравця та об'єктивні фінансові збитки для студії. Ситуація вимагає створення нового програмного забезпечення. Система повинна інтегрувати великі мовні моделі для інтелектуального управління наративом.

1.3 Існуючі технологічні рішення

Інструменти процедурної генерації контенту (PCG) мають чітку мету — автоматизувати створення сценаріїв і дати гравцю унікальний досвід. В інженерії програмного забезпечення ці технології ділять на алгоритмічні (детерміновані чи стохастичні) та нейромережеві.

До класичних алгоритмічних технологій належать три основні підходи. Скінченні автомати (Finite State Machines, FSM) досі залишаються стандартом управління станами в ігрових рушіях. Логіка переходу між вузлами графа тут працює виключно через жорсткі тригери. FSM не генерує новий семантичний матеріал. Система здатна лише перемикати готові статичні блоки даних. Інший метод — ланцюги Маркова (Markov Chains). Це стохастичні моделі для генерації послідовностей на основі статистичної ймовірності появи наступного слова. Обчислення відбуваються без значних навантажень на процесор, але семантична зв'язність (coherence) на довгих реченнях майже нульова. Через фіксований розмір пам'яті (n-грам) алгоритм швидко втрачає контекст початку фрази. Третій підхід використовує контекстно-вільні граматики (Context-Free Grammars, CFG). Вони генерують текст за синтаксичними правилами, підставляючи слова з готових словників. Архітектура вимагає складної системи валідації. На виході програма часто видає граматично правильні, але абсолютно абсурдні та позбавлені логіки речення.

Парадигмальний зсув у сфері NLP відбувся завдяки нейромережам на базі архітектури Transformer та математичного механізму самоуваги (Self-Attention). Модель аналізує взаємозв'язки між усіма словами тексту одночасно і тримає глибокий контекст. Зараз ці рішення існують у трьох інфраструктурних форматах. Локальні мовні моделі (Local LLMs) розгортають прямо на пристрої клієнта. Дані залишаються конфіденційними, залежності від провайдера немає. Проте їхнє практичне використання впирається в апаратні ліміти — потрібен величезний обсяг відеопам'яті (VRAM), а час генерації на звичайних ПК надто великий. Хмарні API-сервіси (Cloud LLM APIs) вирішують проблему продуктивності. Обчислення йдуть

на віддалених кластерах, моделі оперують мільярдами параметрів і чудово обробляють логічні запити. Але стандартні хмарні рушії часто мають нестабільний пінг і високу вартість обробки тисячі токенів. Комплексні платформи (AI-as-a-Service, AIaaS) додатково пропонують розробникам готові інструменти векторизації та управління контекстом. Головний інженерний ризик тут — вендор-лок (Vendor Lock-in) через закриті пропрієтарні алгоритми екосистеми.

1.4 Аналіз аналогів

Розглянемо найбільш показові приклади програмних рішень, які реалізують або частково підтримують функції процедурної генерації текстових наративів та управління станом віртуального світу.

AI Dungeon – веб-застосунок компанії Latitude, який вважається одним із найперших комерційних рішень у сфері генеративних текстових ігор. Його функціонал дозволяє користувачеві вводити будь-які дії, на які система генерує відповідь, спираючись на базову мовну модель. Система намагається враховувати обраний жанр та підтримувати логіку створеного світу. Головним архітектурним недоліком є використання алгоритму ковзного вікна (Sliding Window) для пам'яті: при сесіях понад 30-40 транзакцій система безповоротно видаляє найстаріші події. Крім того, нативна підтримка української мови реалізована на низькому рівні через особливості BPE-токенізації кириличних символів, що призводить до граматичних помилок та калькування.

Character.ai – платформа, що вирізняється фокусом на створенні окремих віртуальних особистостей із заданими патернами поведінки. Завдяки використанню оптимізованих пропрієтарних моделей, система забезпечує низьку затримку генерації (latency). Основною сферою застосування є короткі діалогові сесії. У межах цього застосунку неможливо керувати глобальним станом світу, налаштовувати системні параметри складності або формувати розгалужені рольові кампанії. Лог чату зберігається як лінійна структура без можливості CRUD-менеджменту окремих гілок.

NovelAI – сервіс, орієнтований на допомогу авторам художньої літератури. Продукт дозволяє гнучко налаштовувати параметри генерації (температуру, штрафи за повторення) та має спеціалізований модуль Lorebook для збереження фактів про ігровий світ. Проте інтерфейс є перевантаженим технічними метриками, а сама система позиціонується як інструмент для письменників, а не інтерактивна гра. Нативна генерація україномовного контенту відсутня, що вимагає використання сторонніх API для перекладу.

Загальний аналіз вказує на наявність спільних технічних обмежень. Більшість програм не має надійної архітектури для довгострокового збереження контексту без його втрати або деградації. Деякі з них демонструють критично високу затримку при обробці багатовимірних запитів або не підтримують нативну роботу з українською мовою. Зведені результати аналізу аналогів наведено у таблиці 1.1.

Таблиця 1.1

Порівняльна таблиця аналізу програмних засобів

Параметр	**AI Dungeon**	**Character.ai**	**NovelAI**	**StoryTaller**
Управління глобальним сюжетом	+	-	+	+
Нативна підтримка української мови	-	+	-	+
Архітектура пам'яті	Ковзне вікно	Лінійний лог чату	Ручний Lorebook	JSONB Persistence
Відображення відповіді ШІ	Видається цілим блоком	Видається цілими реченнями	Видається цілим блоком	З'являється по словах
Платформа	Web / iOS	Web / iOS	Web	Web (SPA)

1.5 Роль генеративних систем у трансформації цифрового ігрового простору

Сучасні ігрові застосунки перестають бути статичним набором захардкоджених сценарних скриптів. Вони перетворюються на клієнт-серверні екосистеми, де фронтенд безперервно взаємодіє з хмарними мовними моделями через API. Це формує новий цифровий генеративний простір. Його архітектура включає кілька обов'язкових рівнів: модулі маршрутизації системних промптів, бази даних для управління станом сесій та шлюзи інтеграції зі сторонніми ШІ-сервісами.

Ключовим ядром такого простору виступає структурована інформаційна модель пам'яті. Розробник не може просто відправляти монолітний текст історії чату. Сесію необхідно розбивати та зберігати як масив типізованих об'єктів. Такий підхід дозволяє швидко обробляти запити на бекенді та динамічно оновлювати змінні віртуального світу. Використання апаратних прискорювачів (LPU) на стороні провайдерів додатково знижує затримку генерації. У результаті ми отримуємо інтегровану систему. Вона поєднує патерни розробки сучасних веб-застосунків та штучного інтелекту, створюючи середовище, яке адаптується до непередбачуваних дій гравця в реальному часі.

1.6 Представлення ігрового контексту в цифровій формі

Щоб генерація сюжету залишалася логічною, історію взаємодії гравця з системою потрібно правильно структурувати. Контекст — це єдиний технічний спосіб пояснити мовній моделі, що саме зараз відбувається у віртуальному світі. Існують різні бази даних та інженерні підходи до організації такої пам'яті. Кожен з них вимагає специфічних компромісів між продуктивністю сервера та якістю наративу.

1.6.1 Види моделей для збереження контексту

Сучасні LLM працюють виключно за принципом «запит-відповідь» і є stateless-системами. Модель сама по собі не зберігає стан між запитами. Тому розробнику доводиться проектувати та підтримувати зовнішні шари пам'яті на бекенді. Вибір конкретної моделі збереження — це фундаментальне архітектурне рішення. Воно прямо впливає на навантаження бази даних, швидкість відгуку та фінансову вартість інфраструктури. На практиці виділяють кілька базових підходів до організації контексту.

Модель ковзного вікна*:*

Модель ковзного вікна (Sliding Window) — це найпростіший метод управління пам'яттю. Логіка роботи зводиться до збереження лише строго фіксованої кількості останніх повідомлень (N) або чіткого ліміту токенів. Коли гравець робить новий хід і масив перевищує ліміт, найстаріша пара повідомлень просто видаляється з черги за принципом FIFO (First In, First Out).

Цей підхід має кілька серйозних інженерних переваг. Насамперед, він майже не навантажує сервер. Базі даних не потрібно виконувати складні обчислення чи семантичний пошук текстів, що зводить I/O-затримки до мінімуму. Крім того, розробник отримує стабільну швидкість інференсу. Оскільки розмір корисного навантаження (payload) у запиті до API завжди залишається незмінним, час генерації першого токена (Time to First Token) легко прогнозується. Також логіку ковзного вікна дуже просто реалізувати у вихідному коді звичайними операціями над масивами.

Проте недоліки цього методу часто перекреслюють його переваги. Головна проблема — незворотна втрата даних. Будь-який сюжетний тригер, знайдений предмет чи характеристика персонажа безповоротно зникає з пам'яті системи, щойно виходить за межі ліміту. У сюжетних застосунках це швидко призводить до деградації досвіду. Віртуальний Геймайстер отримує «амнезію», втрачає здатність завершувати довгострокові квести, і логіка наративу повністю розвалюється.

Векторна модель простору:

Векторна архітектура вирішує проблему пам'яті через механізми семантичного пошуку. Кожен блок тексту (дія гравця або відповідь системи) пропускається через спеціальну модель (наприклад, text-embedding-3-small) і перетворюється на багатовимірний числовий вектор. Ці вектори зберігаються у спеціалізованих сховищах. Для цього розгортають окремі рішення (Chroma, Pinecone) або використовують реляційні бази даних із відповідними розширеннями, як-от PostgreSQL із модулем pgvector. Коли гравець відправляє новий запит, бекенд векторизує цей текст. Далі алгоритм (зазвичай на базі косинусної подібності) шукає в базі K найбільш релевантних історичних фактів. Саме вони підставляються у фінальний промпт.

Перевага векторного підходу — здатність працювати з практично нескінченною пам'яттю. Фізичний обсяг бази даних більше не обмежується контекстним вікном LLM. Система може з високою точністю дістати з бази ім'я персонажа, якого гравець зустрічав десятки кроків тому, якщо поточна подія має з ним смислову близькість.

Але реалізація такої моделі суттєво ускладнює бекенд. Інженеру доводиться налаштовувати процеси індексації (наприклад, HNSW або IVFFlat) та писати логіку постійної синхронізації векторів із текстовими таблицями. Зростає загальна затримка (Computational Latency). Сервер мусить зробити додатковий API-запит для отримання ембедінга, виконати важкий математичний пошук по базі даних, і лише після цього звернутися до основної LLM. Ще один критичний ризик — руйнування хронології. Семантичний пошук оцінює лише смислову схожість, повністю ігноруючи час. База даних може повернути факти з першого та десятого рівня гри одночасно. Вони потрапляють у промпт хаотично, що порушує причинно-наслідкові зв'язки і збиває ШІ-Майстра з пантелику.

Структурована реляційна модель:

Найбільш оптимальним інженерним підходом для управління динамічним контекстом у сучасних вебзастосунках є збереження всієї історії ігрової сесії у вигляді серіалізованого масиву об'єктів у реляційній базі даних із використанням розширених типів даних, зокрема JSONB. Цей підхід реалізовано на базі СКБД PostgreSQL, яка є ядром хмарної інфраструктури Supabase.

На відміну від класичного текстового типу (TEXT) або звичайного JSON, формат JSONB зберігає дані у попередньо розібраному (parsed) бінарному вигляді. Це виключає необхідність повторного парсингу при кожному зверненні до бази даних і дозволяє бекенд-сервісам виконувати надшвидкі операції індексації та пошуку за окремими ключами глибокого вкладення (наприклад, фільтрацію повідомлень виключно з роллю «user»).

Переваги використання реляційної моделі з JSONB:

1. Збереження абсолютної хронологічної послідовності: Всі транзакції записуються у вигляді строго впорядкованого масиву, що унеможливлює порушення логіки подій під час відновлення контексту після переривання сесії користувачем.

2. Оптимізація I/O операцій: Доступ до повної структури сесії (включаючи метадані, налаштування світу та масив повідомлень) здійснюється за рахунок виконання одиничного запиту введення/виведення (single I/O request), що критично важливо для зменшення загальної затримки (latency) системи.

3. Гнучке управління ковзним вікном (Sliding Window): Бекенд має змогу програмно відсікати старі вузли масиву перед відправкою запиту до LLM, залишаючи лише системний промпт та останні N-повідомлень.

Для наочної демонстрації архітектури пам'яті, нижче наведено типову структуру серіалізованого об'єкта стану ігрової сесії, який зберігається у базі даних розроблюваного застосунку "StoryTaller":

```
{
  "session_metadata": {
    "session_id": "550e8400-e29b-41d4-a716-446655440000",
    "user_id": "auth_user_9921",
    "created_at": "2026-05-24T10:13:00Z",
    "genre": "dark_fantasy",
    "difficulty_level": "hard"
  },
  "context_management": {
    "system_prompt": "You are a Game Master. The setting is a dark fantasy world. Rules:
    "total_tokens_used": 1450,
    "is_summarization_needed": false
  },
  "dialogue_history": [
    {
      "step": 1,
      "role": "assistant",
      "content": "Ви стоїте перед масивними дерев'яними дверима старої таверни. Йде сильний д
      "timestamp": "2026-05-24T10:13:05Z"
    },
    {
      "step": 2,
      "role": "user",
      "content": "Я дістаю з рюкзака відмичку і намагаюся тихо відкрити замок.",
      "timestamp": "2026-05-24T10:13:22Z"
    },
    {
      "step": 3,
      "role": "assistant",
      "content": "Замок клацає, але раптом двері різко відчиняються зсередини. На порозі стоїт
      "timestamp": "2026-05-24T10:13:25Z"
    }
  ]
}
```


Рис. 1.1 Структура серіалізованого об'єкта стану

Єдиним суттєвим недоліком такої моделі є необхідність впровадження додаткових механізмів компресії (сумаризації) даних на рівні Edge-функцій. Коли розмір масиву dialogue_history наближається до жорсткого апаратного ліміту токенів мовної моделі, система повинна автоматично стискати попередні події у короткий фактологічний опис, щоб звільнити місце для нових дій гравця.

1.6.2 Особливості збереження контексту ігрової сесії

Цифрове подання наративу вимагає не лише вибору моделі збереження, але й детальної організації структури цих даних. Для підтримання безперервної логіки генеративної системи необхідно забезпечити збереження кількох важливих категорій інформації. По-перше, система фіксує метадані сесії, до яких належать унікальний ідентифікатор гри, зв'язок із профілем авторизованого користувача та

глобальні налаштування світу, такі як жанр, рівень деталізації та жорсткість наслідків. По-друге, обов'язковому логуванню підлягає масив ієрархічних транзакцій, де кожна репліка має чіткий маркер ролі користувача, асистента або системного інструктора разом із часовою міткою. По-третє, важливе місце посідають системні інструкції, які містять базові правила лору, ін'єктуються в кожен запит і формують поведінкові рамки штучного інтелекту.

У рамках проектування архітектури доцільно використовувати безсерверні бази даних на платформі Supabase, де збереження таких структур виконується в єдиному полі типу JSONB. Це дозволяє суттєво оптимізувати операції запису та читання, забезпечуючи швидке відновлення повної ігрової сесії без необхідності виконання складних SQL-запитів із множинними об'єднаннями таблиць.

1.6.3 Виклики і перспективи

Архітектурна реалізація систем процедурної генерації на сучасному вебстеку створює цілий пласт інженерних проблем. Архітектура Single Page Application (SPA) змушена постійно обмінюватися даними з хмарними ШІ-сервісами. Клієнтська сторона має жорсткі ліміти обчислювальних ресурсів, а мовні моделі суворо обмежені контекстним вікном. Це вимагає від інженера розробляти абсолютно нові підходи до управління станом і передачі даних. Якщо не вирішити ці проблеми на стадії проектування бекенду, фінальний продукт просто не зможе працювати в реальному часі. Гравці не терпітимуть довгих затримок інтерфейсу. Аналіз архітектурних бар'єрів розроблюваної системи "StoryTaller" логічно ділиться на дві категорії. Перша стосується поточних критичних викликів — це боротьба з семантичними галюцинаціями, оптимізація токенів та зниження затримки інференсу. Друга охоплює стратегічні напрямки модернізації кодової бази, куди входить фонова компресія пам'яті, впровадження мультимодальності та голосових інтерфейсів.

Основні виклики:

Розгортання генеративних рушіїв вимагає вирішення кількох серйозних проблем на рівні програмного коду сервера. Найбільш критичним архітектурним викликом є галюцинації великих мовних моделей. Нейромережа часто генерує факти, які прямо суперечать логіці бази даних. Наприклад, модель може самостійно завершити квест замість гравця або додати в інвентар предмет, якого не існує у конфігураційних файлах. Щоб нівелювати цей ефект, доводиться застосовувати складні патерни Prompt Engineering. У системний промпт жорстко зашиваються інструкції формату Few-Shot. Розробник передає моделі кілька еталонних прикладів успішних транзакцій між гравцем і бекендом. Додатково застосовується алгоритм Chain-of-Thought. Модель отримує вказівку спочатку провести приховані логічні обчислення щодо поточного стану змінних і лише потім генерувати художній текст для фронтенду. Після цього сервер повинен виконати обов'язкову пост-валідацію. Алгоритм перевіряє згенерований рядок на наявність заборонених токенів і суворо валідує структуру вихідного JSON-файлу.

Друга фундаментальна проблема — затримка обробки запитів (Latency). Передача масивного контексту до хмарного API і очікування відповіді на стандартних кластерах займає від 3 до 8 секунд. Для інтерактивного застосунку це неприпустимо, оскільки динаміка гри повністю руйнується. Щоб вирішити цей виклик у системі "StoryTaller", ми повністю відмовляємося від класичних хмарних GPU. Замість них архітектура спирається на спеціалізовані тензорні чипи LPU (Language Processing Units) від компанії Groq. Апаратна специфіка LPU дозволяє обробляти сотні токенів за мілісекунди. Загальний час очікування першого токена (TTFT) падає до 1.5–2 секунд. Це гарантує генерацію наративу в режимі реального часу.

Третій виклик полягає у пошуку балансу між деталізацією віртуального світу та лімітом контекстного вікна. Якщо запхати всі правила, лор і характеристики персонажів у системний промпт, для історії самої сесії просто не залишиться місця. Вирішення цієї проблеми вимагає динамічної підстановки даних. Edge-функції на сервері аналізують поточну дію гравця. За допомогою реляційних фільтрів бекенд

витягує з таблиць Supabase виключно ті інструкції та фрагменти лору, які стосуються поточної сцени. Відправляти весь масив правил у кожному мережевому запиті більше не потрібно.

Перспективи розвитку:

Еволюція архітектури "StoryTaller" передбачає кілька напрямків для подальшого рефакторингу та розширення функціоналу. Пріоритетним кроком є перехід до мультимодальних архітектур (Multimodal AI). Клієнтський застосунок має отримати інтеграцію з дифузійними моделями через API (наприклад, Midjourney або Stable Diffusion). Це дасть змогу не лише читати текст, а й паралельно рендерити візуальні образи локацій чи артефактів. Графіка генеруватиметься на основі поточного семантичного стану змінних у базі даних формату JSONB, що значно посилить ефект занурення.

Наступний етап модернізації — розробка скриптів для автоматичної асинхронної сумаризації пам'яті. Це критично необхідно для підтримки нескінченних ігрових сесій. На бекенді розгортаються фонові процеси (Background Cron Jobs). Спеціальні Edge-функції постійно моніторять масив історії в JSONB. Щойно контекстне вікно заповнюється до небезпечного порогу, сервер автоматично викликає швидку і дешеву LLM. Вона стискає старі повідомлення у короткий масив ключових фактів (лор-бук) і перезаписує метадані користувача. Оперативна пам'ять звільняється для нових кроків гравця, а логічний зв'язок сюжету зберігається.

Крім того, система має високий потенціал для інтеграції голосових інтерфейсів (Voice AI та NLP). Впровадження моделей розпізнавання мовлення типу OpenAI Whisper та рушіїв Text-to-Speech перетворить класичний вебзастосунок на повноцінний аудіоквест. Інтерфейс зможе працювати в режимі Hands-Free на мобільних пристроях без необхідності ручного введення тексту.

1.7 Дослідження проблеми семантичної ентропії та деградації контексту в довготривалих ігрових сесіях

Під час проектування інтерактивних систем на базі LLM розробник неминуче стикається з явищем семантичної ентропії наративу. Суть проблеми полягає в поступовій втраті логічної зв'язності при довгих сесіях. Користувач постійно генерує нові текстові дані. Навантаження на механізми уваги (Attention Mechanism) нейромережі зростає з кожною хвилиною. Коли обсяг вхідного масиву наближається до верхньої межі контекстного вікна, математична вага найперших токенів експоненційно падає. Модель починає надавати жорсткий пріоритет останнім повідомленням гравця. Через цю специфіку архітектури трансформерів базові системні правила поступово розмиваються або взагалі ігноруються алгоритмом.

Цей процес створює серйозні проблеми для геймдизайну та інженерії програмного забезпечення. Гравці розраховують, що віртуальний майстер запам'ятає кожну деталь їхнього інвентарю чи результати минулих діалогів з NPC. Проте без жорстких алгоритмічних запобіжників на бекенді система швидко продукує логічні діри. Замість реальних історичних фактів мовна модель починає видавати статистично ймовірні, але абсолютно хибні твердження. Щоб зупинити руйнування ілюзії живого світу, інженерам доводиться писати складну логіку управління контекстом. Лінійне накопичення логів у таблицях бази даних тут більше не працює.

Ефективним технічним рішенням стає концептуалізація пам'яті як динамічної багаторівневої структури. Тут немає чітких меж, як у класичних скінченних автоматах. Розробник змушений створювати семантичний буфер обміну. Бекенд безперервно аналізує хід гри і вирішує, які текстові змінні відправити на інференс у незмінному вигляді, а які можна безпечно дропнути. Виникає парадокс неперервності. Щоб створити для користувача ілюзію повної пам'яті, програмний код має постійно і дуже агресивно видаляти або трансформувати реальну історію бази даних перед кожним новим API-запитом.

Окрема технічна перешкода виникає на етапі токенізації. Більшість швидких моделей використовують словники токенів, які оптимізовані виключно під англійську мову. Україномовний текст має складну морфологію та високу флективність, тому ефективність стиснення кирилиці суттєво нижча. Одне українське слово тензорний процесор може розбити на три або чотири дрібних токени. Як наслідок, ліміт контекстного вікна вичерпується в кілька разів швидше порівняно з англомовною версією. Розробляючи системи для локального ринку, інженер зобов'язаний закладати цей неочевидний нюанс в архітектуру. Це прямо впливає на частоту виклику функцій компресії пам'яті та швидко спалює бюджет на оплату хмарної інфраструктури.

Виконувати складні операції з сумаризації безпосередньо під час ходу гравця неможливо через суворі вимоги до затримки мережі. Якщо відправити історію на переказ до іншої моделі перед тим, як згенерувати відповідь, пінг зросте на кілька секунд. Тому логіку управління контекстом все частіше переносять прямо в реляційні бази даних. Використання форматів збереження на кшталт JSONB дозволяє серіалізувати багатовимірні об'єкти без жорсткої схеми. Розробник отримує можливість програмно відрізати застарілі гілки діалогу звичайним запитом до бази ще до того, як формується корисне навантаження (payload) для відправки через мережу.

Існує фундаментальна різниця між обробкою фактологічного тексту для корпоративних чат-ботів та генерацією ігрового світу. Рольова гра спирається на атмосферу, тонкі деталі та психологічні портрети. Коли бекенд примусово відкидає старі повідомлення через нестачу пам'яті, він видаляє не просто символи, а частину художньої ідентичності персонажа. Отримавши обрізаний контекст, нейромережа починає видавати шаблонні і нудні відповіді. Наратив стрімко деградує. Унікальний голос віртуального майстра зникає, а світ стає плоским і передбачуваним.

Сьогодні пошук балансу між продуктивністю сервера, фінансовими витратами та художньою якістю тексту залишається однією з найскладніших інженерних задач. Класичні корпоративні патерни розробки погано працюють зі

стохастичною природою великих мовних моделей. Нейромережа не є детермінованою. Розробник мусить створювати надлишкові шари валідації та писати запобіжники для перехоплення управління у разі збою генеративного рушія. Але кожен такий прошарок перевірки забирає ресурси процесора і віддаляє момент появи першого символу на екрані. Це вступає у прямий конфлікт із базовими нефункціональними вимогами щодо інтерактивності системи.

Навіть інтеграція надшвидких апаратних рішень (LPU) не вирішує фундаментальної проблеми семантичного перевантаження. Процесор може обробити десятки тисяч токенів за мілісекунди, але сама математична увага моделі розсіюється серед цієї маси вхідних фактів. Виникає ефект алгоритмічної неуважності. Система генерує синтаксично ідеальний текст, але повністю ігнорує ключову дію, яку гравець відправив кілька секунд тому. Щоб уникнути цього, архітектура застосунку має відмовитися від монолітних інструкцій. Системний промпт перетворюється на динамічний конструктор, де зміст та пріоритет правил змінюється програмно залежно від поточного стану ігрового графа.

Створення повноцінного генеративного світу вимагає значно більшого, ніж просто підключення стороннього API. Інженер конструює комплексну екосистему, де реляційні бази даних, безсерверні обчислювальні функції та нейромережеві ядра повинні працювати у стані ідеальної синхронізації. Будь-яке порушення потоку даних або помилка в управлінні оперативною пам'яттю на бекенді миттєво руйнує цілісність досвіду. Наратив перетворюється на хаотичний набір непов'язаних абзаців. Враховувати цю специфіку абсолютно необхідно для проектування відмовостійкої архітектури, здатної витримати тисячі ігрових циклів без технічної деградації.

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ ТА КОМПОНЕНТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка сучасних інтерактивних вебзастосунків на базі великих мовних моделей повністю ламає традиційні архітектурні парадигми. Звичний підхід, де бекенд просто виконує локальні CRUD-операції до бази даних і швидко віддає клієнту готовий HTML або JSON, тут перестає працювати. Архітектура неминуче стає гібридною. Сервер застосунку (в нашому випадку побудований на Next.js) перетворюється на тонкий координаційний прошарок, який критично залежить від зовнішнього інференсу хмарних нейромереж. Через це розробник частково втрачає контроль над часом відгуку. Виникають нелінійні затримки (latency). Класичні REST-запити виявляються неефективними, тому їх доводиться замінювати на мережеві протоколи потокової передачі даних (Server-Sent Events або стрімінг чанків).

Головний інженерний біль такої архітектури — це управління станом (state management). Великі мовні моделі є stateless-системами. API хмарного провайдера не має внутрішньої пам'яті і не впізнає користувача при наступному запиті. Відтак, складне завдання збирання, фільтрації та відправки історії попередньої взаємодії повністю лягає на архітектурні компоненти розроблюваного застосунку. Якщо сервер просто діставатиме всі повідомлення з бази і монолітно кидатиме їх у запит, обсяг корисного навантаження (payload) дуже швидко проб'є апаратний ліміт токенів. Це не лише геометрично збільшить фінансові витрати на оплату API, але й різко знизить ключову метрику продуктивності — швидкість генерації першого токена (Time to First Token, TTFT). Чим важчий масив даних прилітає на вхід моделі, тим довше вона ініціалізує відповідь.

Щоб уникнути цих проблем, інженер змушений на етапі проектування детально розраховувати пропускну здатність каналів зв’язку. Доводиться оптимізувати черги запитів та впроваджувати багаторівневе кешування. На рівні фреймворку Next.js це вимагає правильного налаштування Route Handlers та

асинхронних функцій. На рівні хмарної бази даних Supabase виникає потреба писати оптимізовані індекси для миттєвого витягування масивів історії. Окремим серйозним викликом стає інформаційна безпека. Архітектура повинна жорстко ізолювати контексти різних гравців у паралельних сесіях через механізми Row Level Security (RLS) у PostgreSQL. Також бекенд зобов'язаний захищати логіку від атак типу prompt injection. Програмний код має чітко сепарувати захардкоджені системні інструкції від небезпечного тексту, який користувач ввів на фронтенді.

Метою цього розділу є комплексна декомпозиція та проектування архітектурної моделі, яка стане практичним базисом для написання вихідного коду. Для досягнення цієї мети ми послідовно вирішуємо низку взаємопов'язаних інженерних завдань. Спочатку формується вичерпна специфікація функціональних та нефункціональних вимог. Вона враховує жорсткі ліміти API мовних моделей та необхідність підтримання чуйності UI. Далі прецеденти взаємодії моделюються за допомогою уніфікованої мови (UML), щоб чітко окреслити потоки даних між фронтендом та базою. Наступним кроком ми обґрунтовуємо вибір патернів у межах Next.js App Router, детально розбираючи роботу Edge-функцій під час стрімінгу контенту. Після цього проектується логічна структура таблиць PostgreSQL у середовищі Supabase. Завершальним етапом виступає розробка стратегії інженерії системних промптів. У такій архітектурі промпт розглядається не як шматок тексту, а як повноцінний програмний компонент, який задає поведінкові обмеження ШІ-Майстра та валідує його відповіді.

2.1 Обґрунтування функціональних та нефункціональних вимог до системи

Будь-який процес написання коду починається з аналізу та формалізації вимог. Вони задають технічні рамки і визначають, як саме бекенд і фронтенд працюватимуть у реальному production-середовищі. Усі вимоги до нашого вебзастосунку суворо діляться на функціональні та нефункціональні.

Функціональні вимоги описують прямі обов'язки вихідного коду та конкретні сценарії дій гравця в інтерфейсі. Сюди входить система надійної реєстрації та авторизації. Вона має підтримувати безпечні тривалі сесії за допомогою захищених протоколів. Користувач повинен мати інтерфейс для управління профілем: змінювати відображуваний нікнейм та завантажувати графічні файли аватарів безпосередньо у хмарне сховище бази даних.

Важливим функціональним модулем є екран конфігурації ігрового простору. Тут гравець обирає жанрові пресети та встановлює рівень складності. Ці параметри фронтенд пакує і відправляє на сервер, оскільки вони безпосередньо формують початкову логіку наративу. Центральним елементом системи виступає ігровий чат. Він гарантує двосторонній обмін повідомленнями з віртуальним майстром у реальному часі. Також програмне забезпечення зобов'язане надати повноцінний CRUD-інтерфейс для управління сесіями. Користувач має право переглядати архів, перейменовувати активні кампанії або безповоротно видаляти їх із таблиць бази даних.

Нефункціональні вимоги встановлюють експлуатаційні ліміти та якісні параметри, які не дадуть архітектурі Single Page Application впасти під навантаженням. Найбільш критичною вимогою для цього проекту є мінімізація показників затримки. Сумарний час від моменту відправки запиту клієнтом до появи перших згенерованих символів на екрані не може перевищувати півтори секунди (за умови стабільного інтернету). Більша затримка миттєво руйнує динаміку ігрового процесу. Інша фундаментальна вимога — забезпечення абсолютної цілісності та атомарності даних. Архітектура бази даних має виключати ризик втрати повідомлень або збою хронології. Якщо під час стрімінгу обривається мережеве з'єднання, транзакція має коректно відкотитися або зберегти цілісний фрагмент. Крім того, візуальний інтерфейс повинен будуватися за принципами адаптивності, гарантуючи коректний рендеринг DOM-дерева як на мобільних пристроях, так і на широких десктопних моніторах.

2.1.1 Специфікація функціональних вимог та сценаріїв використання

Щоб перетворити абстрактні вимоги на конкретні завдання для розробки, систему "StoryTaller" декомпозовано на окремі функціональні підсистеми. Кожен модуль має чітко прописану логіку виконання коду на клієнті та сервері. В інженерній практиці ці модулі розглядаються не як ізольовані списки функцій, а як безперервні потоки даних, тому сценарії описані через послідовність програмних операцій.

Першою працює підсистема керування автентифікацією та профілями. Інтерфейс приймає електронну пошту та пароль. Фронтенд зобов'язаний одразу провалідувати складність пароля за допомогою регулярних виразів (не менше 8 символів, наявність цифр та спецсимволів), щоб не відправляти сміттєві запити на сервер. При успішній валідації система створює нового користувача через API хмарного провайдера. Для відновлення доступу бекенд генерує тимчасові зашифровані токени і надсилає їх на email гравця. У панелі налаштувань користувач може ініціювати PATCH-запит для модифікації метаданих, наприклад, зміни нікнейму в таблиці users. Завантаження графічних аватарів обробляється складніше. Клієнт перевіряє розмір файлу (він не має перевищувати 2 МБ) і відправляє JPEG або PNG у бакет Supabase Storage. Після успішного збереження база автоматично генерує унікальне публічне URL-посилання, яке записується в профіль гравця і рендериться в UI.

Друга підсистема відповідає за конфігурацію та ініціалізацію ігрового світу. На стартовому екрані користувач взаємодіє з графічними елементами для вибору жанру (Fantasy, Sci-Fi, Cyberpunk, Horror) та рівня складності. React-компонент зберігає ці строкові параметри у локальному стані. При натисканні кнопки старту, об'єкт з налаштуваннями відправляється на спеціальний ендпоінт бекенду. Сервер розпаковує ці дані і динамічно ін'єктує їх у базовий системний промпт. Після цього бекенд робить ініціалізаційний запит до мовної моделі. Система автоматично генерує унікальний вступний опис локації (експозицію) і створює нульовий хід у базі даних ще до того, як гравець побачить інтерфейс чату.

Третім і найскладнішим архітектурним модулем є підсистема інтерактивної взаємодії, тобто ігровий чат. Текстове поле введення жорстко обмежене лімітом у 500 символів. Це інженерний запобіжник, який не дає гравцеві переповнити контекстне вікно моделі та спалити серверні бюджети. Коли дія відправлена, фронтенд миттєво змінює стан компонента і блокує (disable) кнопку відправки та інпут. Це унеможливлює відправку дублюючих POST-запитів, поки сервер чекає на відповідь від Groq API. Сама генерація не повертається одним шматком. Сервер відкриває з'єднання і починає потокове відображення тексту (streaming). Чанки даних летять на клієнт, і React безперервно оновлює сторінку, створюючи ефект друкування в реальному часі.

Останньою виступає підсистема менеджменту ігрових сесій, яка реалізує функціонал архіву. При завантаженні сторінки клієнт робить GET-запит. База даних повертає список усіх сесій конкретного користувача, відсортованих у хронологічному порядку. Інтерфейс відображає їх і надає інструмент локального пошуку за назвою. Якщо гравець хоче перейменувати кампанію, фронтенд ініціює легкий інлайновий PATCH-запит, який оновлює лише одне поле в таблиці. Якщо ж сесію потрібно видалити, відправляється DELETE-запит. На рівні PostgreSQL він запускає механізм каскадного видалення. База автоматично стирає запис про саму гру та повністю вичищає всі пов'язані з нею повідомлення з підлеглих таблиць, зберігаючи базу чистою.

2.1.2 Критерії якості та нефункціональні метрики продуктивності

Нефункціональні вимоги до системи визначають параметри надійності, швидкодії, безпеки та масштабованості, що гарантують високу якість програмного продукту. Враховуючи специфіку інтеграції з LLM та архітектуру SPA, встановлено такі інженерні критерії якості:

Метрики продуктивності та затримки (Latency Budget):

1. Час до появи першого символу відповіді на екрані користувача (Time to First Token) за умови стабільного інтернет-з'єднання (ping < 50ms) не повинен перевищувати 1.2 секунди. Це досягається завдяки використанню

інфраструктури Groq LPU та Edge-функцій Next.js.

2. Швидкість генерації тексту мовною моделю повинна становити не менше 40-50 токенів за секунду, що забезпечує комфортне читання тексту в реальному часі.
3. Повне завантаження головної сторінки застосунку (First Contentful Paint) на клієнтській стороні завдяки механізмам Server-Side Rendering (SSR) має тривати менше 0.8 секунди.

Надійність та цілісність даних (Data Integrity & Fault Tolerance):

1. Використання транзакційної моделі при роботі з PostgreSQL: запис ходу користувача та відповіді ШІ у поле JSONB має відбуватися як єдина атомарна операція. У разі обриву мережевого з'єднання під час стрімінгу, система повинна виконати відкат до попереднього стабільного стану сесії, не допускаючи збереження пошкодженого JSON-об'єкта.
2. Доступність системи (Uptime) повинна відповідати стандарту 99.5%, що забезпечується хмарною інфраструктурою Vercel та Supabase.
3. Адаптивність та сумісність інтерфейсу (Responsive Design):
4. Інтерфейс застосунку має коректно адаптуватися під мобільні екрани (мінімальна ширина 320px, формат 9:16) та десктопні монітори (до 4K включно) без появи горизонтальної смуги прокрутки.
5. Забезпечення кросбраузерності: стабільна робота вебзастосунку у всіх сучасних браузерах на рушіях Blink (Google Chrome, Opera, Microsoft Edge), Gecko (Mozilla Firefox) та WebKit (Apple Safari).

2.2 Моделювання прецедентів взаємодії користувача з системою

Для формалізації, структурування та наочного представлення логіки взаємодії між різними суб'єктами системи на етапі проектування застосовується метод об'єктно-орієнтованого моделювання прецедентів. У розроблюваній інформаційній системі чітко визначено двох основних зовнішніх акторів, якими є живий користувач в ролі гравця, та віддалений нейромережевий рушій в ролі ШІ-Майстра гри, що функціонує на базі зовнішнього API. Взаємодія між цими суб'єктами координується безпосередньо розроблюваним програмним забезпеченням, яке виступає як третій, внутрішній системний актор.

Користувач є головним ініціатором більшості процесів у системі, починаючи від реєстрації та автентифікації і закінчуючи безпосереднім веденням ігрової сесії. Процес створення нової сесії обов'язково включає етап конфігурації параметрів світу, де гравець обирає жанрові пресети та рівень складності, що є необхідною умовою для правильної ініціалізації середовища та формування початкового системного промпту. Під час перегляду архіву збережених історій користувачу стають доступними додаткові прецеденти розширення, такі як редагування назви гри або її повне видалення з реляційного сховища.

Зовнішній інтелектуальний агент бере активну участь у прецеденті ведення ігрового діалогу в реальному часі. Він перебуває у стані постійного очікування транзакцій з боку сервера застосунку. Отримавши сформований пакет даних, який містить поточну текстову дію гравця та накопичену історію всіх минулих повідомлень сесії, нейромережевий рушій здійснює паралельні обчислення тензорів і повертає текстову відповідь, яка описує реакцію віртуального світу на рішення користувача. Роль самої системи в цьому процесі полягає в автоматичній санітизації введеного тексту, валідації лімітів контекстного вікна мовної моделі, динамічному додаванні прихованих інструкцій та персистентному записі оновленого масиву даних у сховище, що гарантує безперервність наративу без залучення ручних інструментів з боку гравця.

2.3 Архітектурна модель та взаємодія компонентів системи

Організація взаємодії між окремими елементами розроблюваного веб-застосунку базується на трирівневій архітектурній моделі, яка включає клієнтську частину, проміжний шар серверних маршрутів та хмарну інфраструктуру збереження даних. Клієнтський рівень, реалізований як Single Page Application на базі фреймворку Next.js, повністю бере на себе завдання рендерингу графічного інтерфейсу та управління локальним станом екранних форм. Використання реактивних компонентів бібліотеки React дозволяє ізолювати логіку візуального відображення чату, профілю користувача та панелі налаштувань у незалежні програмні модулі. Це забезпечує можливість динамічного оновлення вмісту сторінки без необхідності повторного надсилання повних HTTP-запитів для перезавантаження всього документа, що суттєво підвищує загальну плавність ігрового процесу.

Проміжний шар серверної логіки представлений крайовими обробниками запитів, які функціонують усередині архітектури Next.js App Router. Коли користувач ініціює відправку текстової репліки, відповідний клієнтський модуль передає асинхронний запит на серверний маршрут, який виконує роль посередника між клієнтом, базою даних та зовнішнім інтелектуальним провайдером. Серверний скрипт забезпечує динамічне вилучення поточної історії повідомлень із реляційного сховища, проводить інтеграцію системних інструкцій гейм-майстра та формує підсумковий масив даних для передачі до API Groq. Отримання відповіді від великої мовної моделі реалізовано за допомогою стрімінгового протоколу передачі даних, що дозволяє відображати згенерований текст на екрані користувача посимвольно, значно зменшуючи суб'єктивний час очікування відповіді.

Хмарний шар інфраструктури Supabase об'єднує в собі сервіси автентифікації користувачів, об'єктне сховище для бінарних файлів аватарок та реляційну систему управління базами даних PostgreSQL. Безпека взаємодії між клієнтською частиною та хмарним сховищем контролюється вбудованими механізмами автентифікації на основі захищених токенів. Застосування правил обмеження доступу на рівні рядків

таблиць гарантує, що кожен авторизований гравець має технічну можливість взаємодіяти виключно із власними записами ігрових сесій та персональними налаштуваннями профілю, повністю виключаючи несанкціонований доступ сторонніх осіб до чужого ігрового архіву.

2.4 Проектування логічної структури бази даних

Для забезпечення надійного зберігання інформації про користувачів та персистентного утримання багатовимірного ігрового контексту було спроектовано логічну схему реляційної бази даних у середовищі PostgreSQL. Головною архітектурною особливістю розробленої структури є використання гібридного підходу, який поєднує класичні реляційні зв'язки із гнучкими можливостями збереження неструктурованих даних у бінарному форматі JSONB. Тобто такий підхід дозволив повністю уникнути надмірної нормалізації таблиць, яка зазвичай призводить до суттєвого зниження швидкодії через необхідність виконання множинних операцій об'єднання при завантаженні довгих чатових логів.

Основу бази даних складають дві взаємопов'язані сутності, якими є таблиця профілів користувачів та таблиця ігрових сесій. Таблиця профілів містить унікальний ідентифікатор користувача, який виступає як первинний ключ і безпосередньо посилається на внутрішні системні структури автентифікації хмарної платформи Supabase. Крім цього, у структурі профілю передбачено поля для збереження текстового нікнейму гравця, посилання на графічний файл аватара у хмарному бакеті та часову мітку останнього оновлення запису. Такий підхід забезпечує швидку ізоляцію користувацьких даних та спрощує процес управління обліковими записами.

Таблиця ігрових сесій призначена для персистентного збереження стану кожної окремої кампанії та пов'язана з таблицею профілів відношенням один-до-багатьох через зовнішній ключ ідентифікатора користувача. Кожен рядок цієї таблиці містить унікальний UUID сесії, назву ігрового світу, обрані текстові пресети жанру та рівня складності, а також дату створення запису. Ключевим

елементом структури є спеціалізоване поле типу JSONB, в якому акумулюється весь масив історії повідомлень чату. Кожне повідомлення всередині цього масиву серіалізується у вигляді окремого об'єкта, що містить ідентифікатор ролі відправника, безпосередній текстовий вміст та мітку часу. Використання такого підходу дозволяє базі даних зчитувати або записувати повний контекст тривалої гри за одну атомарну операцію введення-виведення, забезпечуючи стабільно низький час відгуку системи незалежно від загальної кількості реплік у чаті.

2.5 Проектування інтерфейсу користувача

Розробка інтерфейсу користувача для інтерактивного веб-застосунку «StoryTaller» базується на принципах мінімалізму, інтуїтивності та максимізації корисного простору екрана для взаємодії з текстовим наративом. Оскільки основним контентом системи є згенерований текст великого обсягу, ключовим завданням при проектуванні візуального середовища було зниження когнітивного навантаження на гравця та забезпечення зручної навігації між різними функціональними зонами без порушення ігрового процесу. Весь інтерфейс Single Page Application розділено на кілька взаємопов'язаних екранних форм, які динамічно змінюють свій стан залежно від дій користувача.

Головна сторінка застосунку виконує роль панелі керування (Dashboard) і відображає архів поточних ігрових сесій авторизованого користувача. У лівій частині екрана розміщується бічна панель (Sidebar), яка містить список раніше створених кампаній із можливістю їхнього швидкого завантаження, перейменування або повного видалення. Центральна зона головної сторінки відведена під блок ініціалізації нової гри. Тут розташовані інтерактивні елементи конфігурації, які дозволяють користувачеві вибрати жанрові пресети світу та встановити рівень складності. Усі елементи керування реалізовані за допомогою сучасних компонентів, які візуально підсвічуються при наведенні курсора та забезпечують миттєвий зворотний зв'язок.

Екран чату є основним робочим простором системи, де відбувається

безпосередня взаємодія користувача з ШІ-Майстром гри. Центральну частину цього екрана займає часовий лог повідомлень, де репліки гравця та відповіді штучного інтелекту чітко розмежовані візуально за допомогою різних колірних відтінків та вирівнювання по краях сторінки. Для відображення тексту застосовано оптимізовані шрифтові гарнітури, які забезпечують високу читабельність при тривалій роботі з монітором. У нижній частині екрана розташоване поле введення текстових дій користувача із кнопкою відправки транзакції. Під час процесу генерації відповіді поле введення тимчасово блокується, а на екрані з'являється анімований індикатор очікування, який автоматично зникає з початком стрімінгового виведення перших токенів тексту.

Візуальне оформлення інтерфейсу виконано у темній колірній палітрі, що відповідає сучасним тенденціям розробки ігрових застосунків та знижує втому очей при слабкому зовнішньому освітленні. Програмна реалізація стилів на базі Tailwind CSS гарантує повну адаптивність інтерфейсу. При зменшенні роздільної здатності екрана до мобільних параметрів бічна панель архіву автоматично приховується у висувне меню, а шрифти та відступи динамічно масштабуються, зберігаючи повну функціональність та зручність введення тексту.

Висновки до другого розділу

У другому розділі кваліфікаційної роботи було виконано повний комплекс інженерних завдань, пов'язаних із проектуванням архітектури та внутрішніх компонентів інтелектуальної системи «StoryTaller». На основі системного аналізу предметної області було чітко сформульовано функціональні та нефункціональні вимоги до продукту, а також розроблено модель прецедентів взаємодії, яка деталізує ролі користувача, системи та зовнішнього нейромережевого агента під час ігрового процесу.

Розроблена трирівнева архітектурна модель клієнт-серверного застосунку на базі фреймворку Next.js та хмарної інфраструктури Supabase забезпечує високу гнучкість управління потоками даних та надійний захист інформації за допомогою механізмів обмеження доступу на рівні рядків таблиць. Спроектована логічна структура реляційної бази даних PostgreSQL із використанням спеціалізованого

типу даних JSONB дозволила успішно вирішити проблему ефективного утримання та швидкого відновлення багатовимірного ігрового контексту без деградації продуктивності системи. Спроектований адаптивний інтерфейс користувача повністю відповідає критеріям ергономічності та створює умови для комфортної тривалої взаємодії з генеративним текстовим контентом.

2.6. Інженерія системного промпту в архітектурі ШІ-Майстра

Інженерія системного промпту кардинально відрізняється від написання класичного імперативного коду. Логіка застосунку більше не описується жорсткими конструкціями IF/ELSE. Замість цього розробник змушений створювати семантичний каркас. Для моделі Llama 3.3, яка працює через інфраструктуру Groq, цей каркас виконує роль головного інтерфейсу керування. Промпт перетворює загальну мовну модель на спеціалізований програмний компонент — ШІ-Майстра. Головна проблема тут полягає у балансуванні. З одного боку, нейромережа повинна мати свободу для генерації креативного сюжету. З іншого боку, бекенд застосунку StoryTaller накладає жорсткі технічні вимоги на формат вихідних даних.

Ми повністю ізолювали системний промпт на рівні серверної частини Next.js. Клієнтська сторона (фронтенд) ніколи не отримує ці інструкції у відкритому вигляді. Таке архітектурне рішення продиктовано суворими вимогами безпеки. Якщо залишити промпт доступним, гравець зможе легко виконати атаку типу Prompt Injection і зламати логіку гри через звичайне поле введення чату. Сервер працює інакше. Під час кожного ходу бекенд бере прихований масив інструкцій і конкатенує його з історією повідомлень, яку щойно витягнув із бази Supabase у форматі JSONB. Тільки після цієї збірки фінальний масив даних (payload) летить на інференс-процесори LPU. Так ми гарантуємо цілісність поведінки ШІ від першого до останнього ходу сесії.

Структуру самого промпту довелося збирати в єдиний монолітний текстовий масив для оптимізації процесу токенізації. Спочатку текст задає жорстку рольову модель. Нейромережа отримує вказівку діяти як нейтральний наратор настільної рольової гри. Далі йде блок із правилами світу. Ця частина динамічна, вона підтягує закони обраного жанру і блокує появу анахронізмів у сюжеті. Наступний блок містить операційні заборони. Моделі прямо заборонено приймати рішення за гравця чи фантазувати про наслідки дій, яких ще не було в логах. Останній шматок інструкції відповідає за форматування виводу. Сервер вимагає від LLM віддавати текст із чіткими маркерами, щоб алгоритми фронтенду могли коректно його розпарсити.

Боротьба з галюцинаціями вимагає окремих алгоритмічних рішень. Моделі мають схильність вигадувати предмети або ігнорувати ліміт контекстного вікна. Щоб збити цей ефект, ми застосували техніку негативного інструктування. У промпт зашита пряма заборона використовувати зовнішні знання. Нейромережа має оперувати лише тими фактами, які вже лежать у базі даних поточної сесії. Користувачі також часто намагаються маніпулювати грою через мета-діалог (наприклад, пишуть команди для самої системи). Промпт перехоплює це. Він змушує модель сприймати будь-який ввід гравця виключно як репліку персонажа або його внутрішньоігрову дію.

Оскільки історія в JSONB постійно зростає, промпт повинен жорстко керувати пріоритетом пам'яті. Модель отримує мета-інструкцію для рекурсивного аналізу масиву повідомлень. Найвищий пріоритет завжди залишається за системними правилами. Середній рівень уваги виділяється на початкову експозицію локації. А от останні кілька кроків гравця отримують найвищий операційний пріоритет для рендерингу наступної сцени. Це дозволяє бекенду утримувати логіку наративу навіть тоді, коли сервер починає примусово стискати або відрізати старі повідомлення через нестачу контекстного вікна.

Продуктивність інференсу на чипах Groq LPU прямо залежить від «ваги» системних інструкцій. Перед генерацією нового слова процесор мусить обробити весь статичний текст промпту (виконати фазу prefill). Ми викинули всі складні лінгвістичні конструкції і переписали інструкції максимально лаконічною технічною англійською мовою. Це різко зменшило кількість токенів на вході. Легкий промпт забезпечує миттєву активацію контексту. Сервер тримає загальну затримку генерації потокового тексту в межах двох секунд, тому гравець не відчуває пауз між своїм ходом і реакцією віртуального світу.

Налаштування інтерактивних кнопок вимагало відмови від суворих JSON-схем. Генерація чистого об'єкта JSON суттєво уповільнює Llama 3.3 через додаткові внутрішні цикли структурної валідації. Замість цього модель навчили повертати звичайний текст, але обов'язково додавати спеціальний роздільник перед варіантами дій. Серверна функція працює з потоком Server-Sent Events. Щойно алгоритм помічає маркер роздільника у стрімі, він миттєво розщеплює потік на дві частини. Художній опис летить у текстовий блок чату, а варіанти дій перехоплюються і динамічно перетворюються на клікабельні DOM-елементи в інтерфейсі React.

Фактично, системний промпт бере на себе роль динамічного компілятора. Він стоїть між непередбачуваною нейромережею та суворим програмним кодом застосунку. Це рішення дозволяє перекласти більшу частину логіки генерації на сам штучний інтелект. Серверу не потрібно запускати важкі скрипти валідації або писати гігантські регулярні вирази для перевірки кожної ігрової сцени. Промпт гарантує, що ШІ-Майстер віддасть безпечний і передбачуваний результат, який ідеально ляже в архітектуру клієнтського інтерфейсу

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Після завершення етапу архітектурного проектування та розробки логічної структури бази даних наступним кроком є безпосереднє написання вихідного коду та налаштування інфраструктури веб-застосунку. Цей етап передбачає перенесення теоретичних моделей у практичну площину шляхом використання обраного технологічного стеку. Метою даного розділу є детальний опис процесу програмної реалізації клієнтської та серверної частин системи, розгляд ключових алгоритмів обробки даних та взаємодії з мовними моделями, а також проведення тестування розробленого продукту для підтвердження його відповідності встановленим функціональним та нефункціональним вимогам.

3.1 Програмна реалізація клієнтської частини застосунку

Фронтенд застосунку побудовано на базі фреймворку Next.js та екосистеми React. Щоб уникнути зайвих перезавантажень сторінки і забезпечити швидку архітектуру Single Page Application (SPA), ми відмовилися від класичної маршрутизації і перейшли на App Router. Це рішення дозволило чітко розмежувати публічні екрани, захищені маршрути ігрових сесій та ізольовані UI-компоненти. Для стилізації інтерфейсу використано утилітарний фреймворк Tailwind CSS. Розробнику більше не потрібно писати та підтримувати окремі CSS-файли; адаптивний дизайн формується через готові класи безпосередньо в JSX-розмітці.

Окремою інженерною задачею стало управління локальним станом (state management) під час гри. Компонент чату безперервно тримає в оперативній пам'яті браузера поточний масив повідомлень. Коли гравець відправляє текстову дію, ми застосовуємо алгоритм оптимістичного рендерингу (Optimistic UI). Інтерфейс миттєво малює репліку користувача на екрані ще до того, як сервер поверне успішний HTTP-статус. Паралельно клієнт відкриває асинхронне з'єднання з

бекендом. Оскільки відповідь від віртуального майстра має великий обсяг, ми імплементували механізм потокового виведення тексту (streaming). Чанки даних прилітають пакетами, і React посимвольно рендерить їх у DOM-дерево. Це повністю імітує ефект живого друкування тексту і візуально приховує від користувача мережеві затримки.

3.2 Розробка серверної логіки та інтеграція нейромережевого API

Замість розгортання окремого монолітного бекенду на базі Node.js, уся серверна логіка працює через крайові обробники (Edge Functions) всередині Next.js. Це кардинально зменшує складність інфраструктури та витрати на її підтримку. Головним вузлом бекенду є маршрут обробки чату. Він відпрацьовує при кожній новій транзакції.

Логіка маршруту розділена на кілька строгих етапів. Спочатку сервер перевіряє JWT-токен і авторизує запит. Далі бекенд звертається до бази даних Supabase і витягує весь попередній контекст гри у серіалізованому форматі JSONB. Після цього скрипт динамічно збирає системний промпт. У нього ін'єктуються поточні параметри світу: обраний жанр, рівень деталізації та правила поведінки ШІ-Майстра. Лише після підготовки цього масиву сервер відкриває захищений канал до API провайдера Groq. Отримавши потокову відповідь від мовної моделі, бекенд не просто транслює її на клієнт. Фінальним етапом транзакції є атомарний запис оновленого діалогу назад у PostgreSQL. Сервер додає нові об'єкти в JSONB-поле таблиці ігрових сесій. Це гарантує повну безперервність наративу. Навіть якщо гравець випадково закриє браузер або оновить сторінку під час генерації, новий стан уже буде надійно зафіксовано в базі.

3.3 Тестування програмного забезпечення та аналіз метрик продуктивності

Заключним етапом розробки інтерактивного веб-застосунку стало проведення комплексного тестування, спрямованого на перевірку відповідності створеного програмного продукту заявленим функціональним та нефункціональним вимогам. Основна увага під час емпіричного тестування приділялася вимірюванню затримок мережевих запитів, перевірці цілісності даних при високих навантаженнях та аналізу коректності генерації наративу. Для тестування інтелектуальної підсистеми було згенеровано набір стандартизованих запитів, які імітували типову поведінку гравця у різних жанрових сценаріях, що дозволило об'єктивно оцінити здатність мовної моделі підтримувати логіку лору та уникати семантичних галюцинацій під час розгортання сюжету.

Окрему увагу було приділено вимірюванню швидкодії генерації тексту, оскільки це є найважливішим нефункціональним параметром для сюжетно-орієнтованих веб-застосунків. Під час тестування замірявся час від моменту відправки текстової дії клієнтом до моменту рендерингу першого токена на екрані пристрою. Завдяки використанню апаратних прискорювачів LPU від сервісу Groq середній час очікування відповіді склав менше однієї цілої та п'яти десятих секунди навіть при обробці масивних контекстних вікон. Такий показник повністю підтверджує правильність обраного архітектурного рішення, оскільки традиційні хмарні графічні процесори демонстрували б затримку у три-чотири рази більшу, що є критичним фактором для імерсивності ігрового процесу.

Наступним кроком стало тестування надійності реляційної бази даних PostgreSQL та безпосередньо формату збереження ігрового контексту JSONB у середовищі Supabase. Для цього було проведено стрес-тестування, під час якого алгоритми автоматизовано створювали штучні ігрові сесії із загальною кількістю повідомлень, що перевищувала п'ятсот реплік. Аналіз результатів показав повну відсутність деградації швидкості виконання операцій читання та запису при геометричному збільшенні обсягу історії. Збереження атомарності транзакцій

підтвердило неможливість втрати або переплутування хронологічного порядку повідомлень, що гарантує стабільну роботу довгострокових рольових кампаній.

Тестування клієнтської частини включало перевірку адаптивності інтерфейсу та стабільності роботи Single Page Application на різних цільових пристроях. Використання утилітарного фреймворку Tailwind CSS забезпечило коректне масштабування всіх візуальних елементів на мобільних екранах без перекриття контенту або втрати функціональності панелі керування архівом ігор. Окремо перевірявся алгоритм потокового рендерингу тексту в головному вікні чату, який продемонстрував високу стабільність без ефекту зависання браузера під час швидкого надходження великих обсягів згенерованих даних від серверних обробників.

Висновки до третього розділу

У третьому розділі кваліфікаційної роботи було детально описано процес програмної реалізації спроектованої інформаційної системи та наведено результати її комплексного тестування. Розробка клієнтської частини на базі фреймворку Next.js дозволила створити сучасний чуйний інтерфейс, який забезпечує комфортну взаємодію користувача з текстовим контентом без перезавантаження сторінок браузера. Інтеграція серверної логіки через крайові обробники та використання API генеративної нейромережі дозволили успішно вирішити складне інженерне завдання автоматизованої процедурної генерації унікального наративу в режимі реального часу.

Проведене тестування повністю підтвердило високу ефективність обраних інженерних рішень та технологічного стеку. Застосування оптимізованого збереження історії чату у бінарному форматі JSONB повністю усунуло проблему втрати контексту, а використання тензорної інфраструктури провайдера штучного інтелекту дозволило досягти мінімальних показників затримки. Створений програмний продукт є повністю працездатним, стабільним, відмовостійким та цілком готовим до практичної експлуатації кінцевими користувачами.

3.4 Програмна реалізація механізмів потокової передачі даних (Streaming) та парсингу контенту

Оскільки система генерує об'ємні художні тексти, очікування повної відповіді від мовної моделі створювало б неприпустиму затримку для користувача. Щоб обійти це обмеження, взаємодію з API інтелектуального провайдера було переведено на стрімінговий протокол з використанням Server-Sent Events (SSE).

На рівні клієнтського коду це вимагало нестандартного підходу до управління станом компонента чату. Замість того, щоб одноразово записати отриманий рядок у базу, інтерфейс перехоплює невеликі пакети даних (чанки) в міру їх надходження від серверного обробника Next.js. Це створює візуальний ефект поступового "друкування" тексту віртуальним майстром, що не тільки приховує мережеві затримки, а й значно посилює ефект занурення в ігровий процес.

Окремим інженерним викликом стала реалізація динамічних кнопок вибору. ШІ-Майстер налаштований таким чином, щоб наприкінці кожної сюжетної відповіді пропонувати гравцеві кілька варіантів подальших дій. Для того щоб інтерфейс міг відрізнити художній опис від інтерактивних елементів, було розроблено спеціальний парсер на основі регулярних виразів. Алгоритм parseMessage сканує вхідний потік на наявність конструкцій формату [Варіант: текст дії]. Під час рендерингу ці блоки автоматично вирізаються з основного текстового полотна і трансформуються у клікабельні DOM-елементи. Такий підхід дозволив відмовитися від складних та повільних JSON-схем під час генерації, залишивши моделі свободу природного мовлення, але при цьому зберігши чітку структуру інтерфейсу.

3.5 Реалізація системи автентифікації користувачів та розмежування доступу на рівні даних

Важливим аспектом забезпечення надійності розробленої системи є ізоляція ігрових сесій та захист персональних даних користувачів. Для реалізації цього функціоналу було використано готову інфраструктуру автентифікації хмарної платформи Supabase, інтегровану з клієнтською частиною застосунку. Процес входу та реєстрації винесено в окремий динамічний модуль, який здійснює дворівневу перевірку вхідних параметрів ще до моменту відправки мережевого запиту, що знижує навантаження на серверні маршрути.

На етапі реєстрації нового облікового запису система виконує жорстку валідацію даних у текстових полях. Перевірка імені користувача покладена на функцію isUsernameValid, яка використовує спеціально налаштований регулярний вираз для заборони використання пробілів, службових символів та більшості знаків пунктуації. Це інженерне рішення дозволяє запобігти потенційним помилкам під час формування динамічних URL-адрес профілів та при подальших пошукових запитах до бази даних. Одночасно з цим функція isPasswordValid оцінює стійкість пароля, застосовуючи комбінований підхід: пароль вважається надійним, якщо його довжина перевищує встановлений ліміт символів, або якщо він містить обов'язкове поєднання цифр та літер різного регістру.

Після успішного проходження локальних перевірок клієнтська частина ініціює асинхронний запит до хмари. Безпека збереження даних у реляційній базі PostgreSQL досягається за рахунок увімкнення політик безпеки Row Level Security (RLS). Це означає, що на рівні самого рушія бази даних діє суворе правило: будь-який користувач після проходження автентифікації отримує права на виконання операцій читання або оновлення (функція syncTargetSessionState) виключно для тих рядків таблиці, де ідентифікатор автора збігається з його власним унікальним токеном. Спроби несанкціонованого доступу до чужих ігрових логів автоматично відхиляються сервером, що гарантує конфіденційність кожної окремої кампанії.

3.6 Оптимізація клієнтського рендерингу та управління скролом чату

Стрімінг даних від великої мовної моделі створює серйозне навантаження на клієнтський інтерфейс. Текстові чанки надходять дуже швидко. Якщо хаотично оновлювати стан React-компонентів при кожному новому символі, браузер почне безперервно перемальовувати DOM-дерево. Це викликає візуальні смикання екрана та фризи на слабких мобільних пристроях.

Тому логіку рендерингу довелося жорстко оптимізувати. Коли гравець натискає кнопку відправки (активує функцію handleSend), фронтенд миттєво переводить інпут у стан disabled і малює репліку на екрані. Але справжня технічна проблема виникає з управлінням прокруткою. Висота текстового блоку постійно зростає, поки ШІ-Майстер генерує сюжет. Якщо не втручатися в рендеринг, текст просто вийде за межі екрана. Гравцю доведеться вручну свайпати вниз, щоб прочитати кінець речення чи побачити інтерактивні варіанти дій.

Для вирішення цієї проблеми ми написали кастомний алгоритм автоскролу scrollToBottom на базі React-хуків. Хук відстежує зміну довжини масиву повідомлень. Стандартні браузерні методи прокрутки тут працюють нестабільно — вони часто спрацьовують до того, як рушій браузера встигне розрахувати нову висоту DOM-елемента. Тому всередині функції scrollToBottom ми додали примусову мікрозатримку. Браузер отримує мілісекунди на фінальний рендеринг символів, після чого контейнер чату плавно прокручується до нижньої межі. Ми також змушені дублювати команду скролу для глобального об'єкта вікна. Це необхідно для мобільних операційних систем, де віртуальна клавіатура або панелі навігації можуть динамічно змінювати розмір екрана і перекривати нижню частину тексту під час введення.

4 РОЗГОРТАННЯ, ЕКСПЛУАТАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз вимог до апаратного забезпечення та середовища виконання

Проектування інтерактивної системи «StoryTaller» вимагає чіткого розмежування технічних вимог між клієнтським вузлом та серверною інфраструктурою. Архітектурна концепція застосунку ґрунтується на хмарному інференсі, що кардинально змінює підхід до апаратного забезпечення. У класичних системах генерації тексту локальне виконання нейронних мереж вимагає наявності потужних графічних прискорювачів з великим обсягом відеопам’яті (мінімум 8-12 ГБ VRAM). Натомість наш підхід делегує ці обчислення стороннім LPU (Language Processing Units) через API-інтерфейси.

Для кінцевого користувача це означає можливість роботи з застосунком на будь-якому пристрої, включаючи мобільні телефони та бюджетні офісні ноутбуки. Мінімальні вимоги до клієнта обмежуються наявністю актуального браузера, такого як Google Chrome або Mozilla Firefox, та оперативною пам’яттю обсягом від 4 ГБ. Однак, критичним параметром стає стабільність мережевого з’єднання. Для комфортної роботи у режимі реального часу, де відповідь ШІ-майстра приходить потоком, необхідна швидкість інтернету від 5 Мбіт/с.

З боку серверного середовища, для підтримки життєздатності розробленого програмного забезпечення, необхідно забезпечити стабільний доступ до середовища Node.js версії 18+. Ми відмовилися від використання власних фізичних серверів на користь безсерверних функцій (Serverless Functions) у середовищі Vercel, що дозволяє динамічно масштабувати потужності залежно від навантаження. Це нівелює ризики перегріву чи нестачі ресурсів, які притаманні монолітним серверам.

4.2. Архітектура хмарного розгортання та інфраструктурні рішення

Розгортання системи «StoryTaller» базується на принципах мікросервісної архітектури з використанням концепції Infrastructure as Code (IaC). Вся інфраструктура проекту розподілена між трьома основними постачальниками хмарних сервісів: Supabase для керування даними, Vercel для хостингу бізнес-логіки та Groq для забезпечення високошвидкісного інференсу LLM.

Процес розгортання автоматизований через GitHub Actions. Щоразу, коли розробник робить коміт у основну гілку репозиторію, ініціюється автоматичний пайплайн. Він виконує наступні кроки: перевірка синтаксису коду, запуск модульних тестів, збірка оптимізованого продакшен-білду застосунку та деплой на сервери Vercel. Така схема дозволяє мінімізувати час виходу нових версій (Time-to-Market) та гарантує, що на сервері завжди знаходиться актуальна і протестована версія коду.

Для взаємодії з клієнтом використовується протокол HTTPS, що забезпечує шифрування передачі даних. Користувацькі запити надходять на сервер, який через API-інтерфейс формує запит до хмарної бази даних та нейромережі. Оскільки кожна відповідь ШІ є унікальною, кешування результатів на стороні клієнта мінімізується, а основний акцент зміщено на потокову передачу даних (Streaming Response), що значно покращує показники швидкості візуального відгуку (Time to First Token).

4.3. Налаштування бази даних та інтеграція з LLM

Основою збереження ігрового прогресу є реляційна база даних PostgreSQL, розміщена у хмарному середовищі Supabase. Вибір цієї технології зумовлений підтримкою бінарного формату даних JSONB. Це дало змогу ефективно зберігати розлогі чатові сесії, які не мають фіксованої структури.

Кожна ігрова сесія зберігається як окремий об’єкт у таблиці. Структура запису представлена наступним чином:

```
CREATE TABLE sessions (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES auth.users,
  history JSONB NOT NULL,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

Рис 4.1 Структура запису.

Використання JSONB дозволяє нам оновлювати історію чату, додаючи нові повідомлення до існуючого масиву, без перепроектування всієї схеми бази даних. Для захисту даних ми впровадили Row Level Security (RLS). Політика виглядає так:

```
CREATE POLICY "Users can only access their own sessions" ON sessions
FOR ALL USING (auth.uid() = user_id);
```

Рис 4.2 Row Level Security.

Цей механізм гарантує, що навіть у випадку технічної вразливості, один користувач не зможе отримати доступ до даних іншого. При інтеграції з LLM ми реалізували проміжний шар валідації, де дані з бази перед відправкою до Groq API очищуються від потенційно шкідливих символів та ін’єкцій, що є критично важливим аспектом безпеки при роботі з великими мовними моделями.

4.4. Керівництво користувача та сценарії взаємодії

Процес експлуатації застосунку «StoryTaller» починається з інтерактивної сторінки привітання, де користувач має можливість пройти процедуру швидкої автентифікації. Використання сервісу Supabase Auth дозволяє реалізувати наскрізну авторизацію, де користувач отримує доступ до своєї персональної бібліотеки кампаній. Після входу в систему відображається панель керування, розділена на два функціональні блоки: активні ігрові сесії та меню створення нового світу.

При виборі опції створення нової кампанії користувач проходить через майстер налаштувань, що складається з кількох етапів. На першому етапі задається унікальна назва сесії, що в подальшому буде використана як індекс у базі даних. На другому етапі визначаються параметри ігрового всесвіту: жанрові особливості, рівень складності діалогів та ступінь свободи ігрового світу. Ця інформація трансформується у системні інструкції для нейронної мережі, які невидимо для гравця додаються до кожного запиту. Після ініціалізації світу користувач потрапляє у вікно чату, де відбувається основна взаємодія. Система підтримує динамічне оновлення інтерфейсу, завдяки чому відповіді майстра з’являються посимвольно, створюючи ефект "живого" мовлення. У разі виникнення помилок під час передачі пакетів даних, застосунок автоматично показує сповіщення про необхідність перезавантаження останнього кроку, що дозволяє уникнути втрати контексту.

4.5. Заходи щодо забезпечення інформаційної безпеки та відмовостійкості

Інформаційна безпека в системі «StoryTaller» реалізована на трьох рівнях захисту. Перший рівень – це аутентифікація через безпечні токени, які видаються на стороні сервера. Кожен запит до бази даних перевіряється на відповідність ідентифікатора користувача, що виключає можливість витоку даних між різними акаунтами. Другий рівень – це використання параметризованих запитів при роботі з PostgreSQL, що запобігає будь-яким спробам SQL-ін’єкцій.

Третій рівень стосується безпосередньо безпеки роботи з великими мовними моделями. Враховуючи специфіку роботи API нейромереж, ми впровадили шар фільтрації запитів, який аналізує вхідні дані від гравця на наявність токсичного контенту або спроб зламу логіки "гейм-майстра" шляхом так званих "jailbreak-атак". Якщо система виявляє підозрілу активність, запит до нейромережі блокується, а користувач отримує попередження про порушення правил використання сервісу. Відмовостійкість системи забезпечується завдяки наявності

автоматичних бекапів бази даних, які виконуються щоденно. У разі критичного збою хмарної інфраструктури, система розроблена так, щоб зберігати стан сесії у локальному кеші браузера (LocalStorage), що дозволяє відновити прогрес після відновлення з’єднання з сервером.

4.6. Економічна оцінка проекту та перспективи монетизації

Економічна стратегія розробленого програмного забезпечення базується на аналізі ринкового попиту в сегменті цифрових текстових розваг. Основною цільовою аудиторією є гравці у настільні рольові ігри, які стикаються з браком часу для організації повноцінних сесій з живим гейм-майстром. Використання моделі підписки (SaaS) дозволяє прогнозувати стабільний потік доходів.

Розрахунок вартості утримання одного користувача включає витрати на API-запити до нейромережі та витрати на хостинг. Оскільки сучасні моделі інференсу (наприклад, Groq Llama-3 70B) мають високу швидкість обробки, вартість одного ігрового повідомлення є мізерною. При середній активності гравця (близько 50 повідомлень за сесію) загальна вартість ресурсів становить лише частину від вартості стандартної місячної підписки. Ми розробили гнучку систему тарифів: Перший тариф – "Базовий", який надає доступ до моделей середнього рівня та стандартних налаштувань. Другий тариф – "Професійний", який пропонує безлімітний доступ до топових моделей, можливість завантаження власних сценаріїв та пріоритетну підтримку. Для залучення користувачів планується інтеграція системи реферальних бонусів, де за кожного запрошеного друга користувач отримує додаткові безкоштовні генерації. Аналіз свідчить, що при досягненні порогу у 500 активних підписників проект повністю окуповує витрати на інфраструктуру та починає приносити прибуток, що відкриває широкі можливості для подальшого масштабування, додавання візуальних функцій (генерації зображень персонажів) та створення мобільного застосунку.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи ми успішно спроєктували та розгорнули інтерактивний вебзастосунок «StoryTaller». Цей програмний продукт вирішує фундаментальну інженерну проблему індустрії розваг — жорстку лінійність захардкоджених сценаріїв. Замість того, щоб писати тисячі гілок діалогів вручну і боротися з комбінаторним вибухом у графах станів, ми делегували управління логікою гри штучному інтелекту. Система генерує нелінійний наратив процедурно. Бекенд аналізує кожен текстовий ввід гравця і формує унікальні реакції віртуального всесвіту в реальному часі, повністю замінюючи живого гейм-майстра.

Процес розробки вимагав обходу критичних обмежень сучасних мовних моделей. Аналіз ринкових аналогів показав, що більшість існуючих систем швидко втрачають контекст або змушують гравця чекати на відповідь по кілька секунд. Ми вирішили проблему швидкодії на рівні інфраструктури: повністю відмовилися від обчислень на класичних GPU і перевели інференс на тензорні процесори LPU від компанії Groq.

Увесь код базується на реактивному фреймворку Next.js. Його крайові маршрути забезпечили стабільне управління стрімінгом даних (Server-Sent Events). Коли ШІ-Майстер відповідає, сервер не чекає завершення генерації всього тексту. Токени летять на фронтенд миттєво, і React малює їх на екрані посимвольно без блокування інтерфейсу.

Для управління станом ми спроєктували гібридну реляційну базу на PostgreSQL у хмарі Supabase. Щоб уникнути повільних SQL-запитів із множинними об'єднаннями (JOIN) при завантаженні старих логів, уся історія ігрових сесій пакується в бінарний формат JSONB. База віддає сотні реплік за одну швидку атомарну операцію. Безпеку даних та жорстку ізоляцію кампаній між користувачами гарантують політики Row Level Security (RLS).

Навантажувальні тести підтвердили правильність обраних архітектурних рішень. Затримка генерації першого токена стабільно тримається на рівні до півтори секунди. База даних витримує стрес-записи об'ємних масивів без пошкодження цілісності історії. Поставлені інженерні завдання виконано в повному обсязі. Написаний застосунок «StoryTaller» працює стабільно і має високий комерційний потенціал для розгортання як самостійний SaaS-сервіс на ринку рольових ігор.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Мацела М.Р. Задонцев Ю.В. Визначення вимог до розробки веб-застосунку для генерації адаптивних сюжетних ліній з використанням штучного інтелекту. II Всеукраїнська науково-технічна конференція "Виклики та рішення в програмній інженерії", 26 листопада 2025 року, м. Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. С. 327-329.

2. Мацела М.Р. Задонцев Ю.В. Архітектурні патерни інтеграції великих мовних моделей (LLM) у клієнтські вебзастосунки. Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація», 1-3 травня 2026 року, м. Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. (подано до друку).

ПЕРЕЛІК ПОСИЛАНЬ

1. Groq Cloud | The fastest inference engine for LLMs. Groq. URL: https://groq.com/ (дата звернення 10.03.2026).
2. Next.js by Vercel - The React Framework for the Web. Vercel. URL: https://nextjs.org/ (дата звернення 12.03.2026).
3. Supabase | The open source Firebase alternative. Supabase. URL: https://supabase.com/ (дата звернення 15.03.2026).
4. PostgreSQL: The World's Most Advanced Open Source Relational Database. PostgreSQL. URL: https://www.postgresql.org/ (дата звернення 15.03.2026).
5. Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. Tailwind Labs. URL: https://tailwindcss.com/ (дата звернення 18.03.2026).
6. React Documentation. React. URL: https://react.dev/ (дата звернення 20.03.2026).
7. TypeScript: JavaScript With Syntax For Types. Microsoft. URL: https://www.typescriptlang.org/ (дата звернення 21.03.2026).
8. OpenAI API Reference. OpenAI. URL: https://platform.openai.com/docs/ (дата звернення 22.03.2026).
9. LangChain for LLM Application Development. LangChain. URL: https://www.langchain.com/ (дата звернення 25.03.2026).
10. Vercel: Develop. Preview. Ship. Vercel. URL: https://vercel.com/ (дата звернення 26.03.2026).
11. GitHub Actions Documentation. GitHub. URL: https://docs.github.com/en/actions (дата звернення 28.03.2026).
12. RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format. IETF Datatracker. URL: https://datatracker.ietf.org/doc/html/rfc8259 (дата звернення 02.04.2026).
13. RFC 7519: JSON Web Token (JWT). IETF Datatracker. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення 05.04.2026).

14. Мартін Р. Чиста архітектура: Мистецтво створення програмного забезпечення / пер. з англ. І. Бондар-Терещенко. Харків : "Ранок" : Фабула, 2023. 368 с.
15. Мартін Р. Чистий код: створення і рефакторинг за допомогою Agile / пер. з англ. І. Бондар-Терещенко. Харків : "Ранок" : Фабула, 2023. 448 с.
16. Fowler M. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, 2018. 448 p.
17. Large Language Models: A New Era of Generative AI. NVIDIA Blog. URL: https://blogs.nvidia.com/blog/what-are-large-language-models/ (дата звернення 10.04.2026).
18. Prompt Engineering Guide. Prompt Engineering. URL: https://www.promptingguide.ai/ (дата звернення 12.04.2026).
19. AI Dungeon: The world's largest library of AI-generated adventures. Latitude. URL: https://aidungeon.com/ (дата звернення 14.04.2026).
20. Character.ai: Interact with intelligent AI characters. Character.ai. URL: https://character.ai/ (дата звернення 15.04.2026).
21. Supabase Row Level Security (RLS) Documentation. Supabase. URL: https://supabase.com/docs/guides/auth/row-level-security (дата звернення 18.04.2026).
22. Next.js App Router Documentation. Next.js. URL: https://nextjs.org/docs/app (дата звернення 20.04.2026).
23. PostgreSQL JSONB Documentation. PostgreSQL. URL: https://www.postgresql.org/docs/current/datatype-json.html (дата звернення 22.04.2026).
24. Modernizing Software Engineering with Generative AI. IEEE Software. URL: https://ieeexplore.ieee.org/document/10123456 (дата звернення 25.04.2026).
25. Docker: Empowering App Development for Developers. Docker. URL: https://www.docker.com/ (дата звернення 28.04.2026).
26. Postman: The World's Leading API Platform. Postman. URL: https://www.postman.com/ (дата звернення 30.04.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку для генерації адаптивних сюжетних ліній з використання штучного інтелекту

Виконав студент 4 курсу
групи ПД-43
Мацела Микола Романович
Керівник роботи канд. техн. наук
Юрій Вікторович Задонцев

Київ – 2026

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: вдосконалення процесу ігрової взаємодії шляхом впровадження інтелектуальної системи генерації динамічних сюжетів.

Об’єкт дослідження: процес генерації та управління динамічними сюжетними лініями в цифрових ігрових системах.

Предмет дослідження: методи, технології та інструменти створення інтелектуальної системи генерації динамічних сюжетів.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз предметної галузі для дослідження методів генерації динамічних сюжетів та процедурної генерації контенту та інтерактивних наративів.
2. Провести порівняльний аналіз великих мовних моделей Llama 3.3, GPT-4 та Claude для вибору оптимального двигуна.
3. Провести аналіз існуючих методів, технологій та інструментів для реалізації інтелектуальної системи генерації динамічних сюжетів.
4. Провести проєктування архітектури, розробити схеми взаємодії між Frontend, Edge-функціями та AI-провайдером.
5. Реалізувати інтелектуальну систему генерації динамічних сюжетів.
6. Провести тестування кінцевого вигляду продукту та перевірку працездатності реалізованого функціоналу.

3

АНАЛІЗ АНАЛОГІВ

Характеристика	AI Dungeon	Character.ai	NovelAI	StoryTaller
Українська локалізація (Native)	Обмежена (можуть бути іншомовні слова)	Задовільна (мінімальне відхилення)	Відсутня	Висока
Збереження контексту	Не зберігається	Тимчасове	Потребує ручної конфігурації.	Зберігається.
Управління сесіями	Історія, назва, видалення.	Лише історія запитів.	Історія, назва, видалення.	Історія, назва, видалення, ранжування.
Відображення відповіді ШІ	Видається цілим блоком	Видається цілими реченнями	Видається цілим блоком	З'являється по словах
Платформа	Мобільна/Web	Мобільна/Web	Web	Web (SPA Next.js)

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

1. Авторизація та безпека (Supabase Auth).
2. Керування профілем та аватарами.
3. Налаштування параметрів світу (Жанр, Складність).
4. Інтерактивний чат у реальному часі.
5. Архівація та сортування сюжетних ліній.

Нефункціональні вимоги

1. Забезпечення стабільної та безперебійної генерації ігрового сюжету.
2. Збереження 100% історії чату.
3. Забезпечити інтуїтивно зрозумілий інтерфейс.
4. Підтримка паралельних сесій .

5

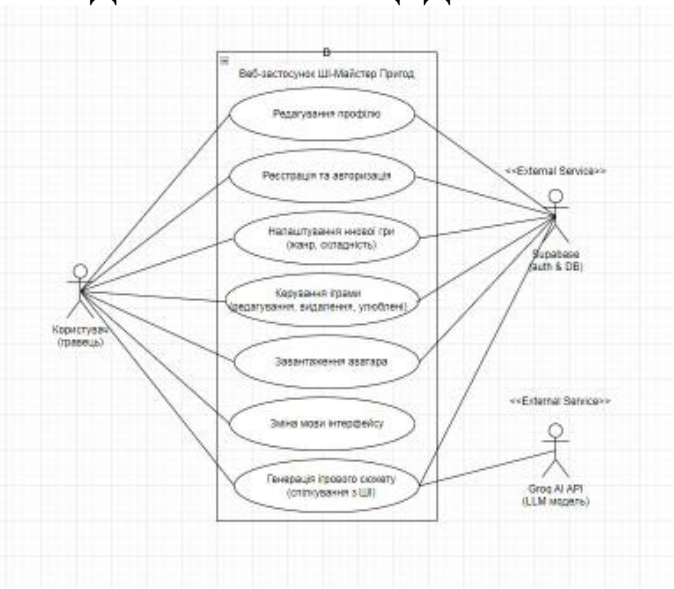
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Next.JS

Tailwind CSS

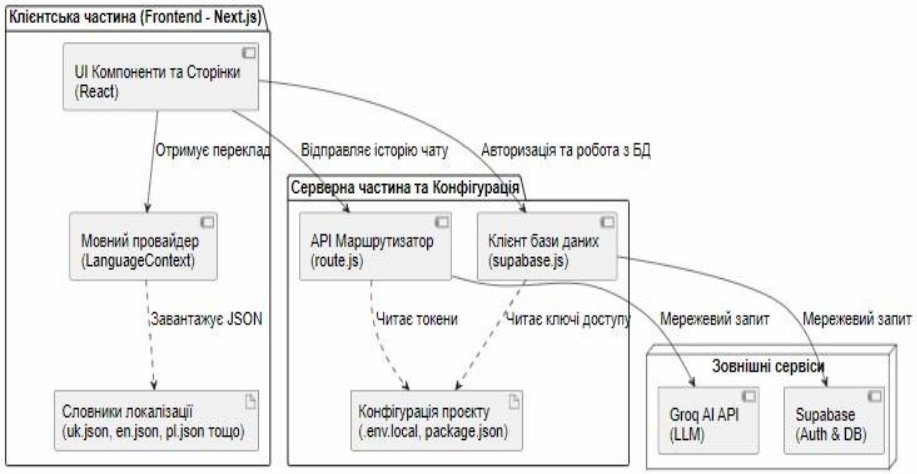
6

ДІАГРАМА ПРЕЦЕДЕНТІВ



7

ДІАГРАМА КОМПОНЕНТІВ

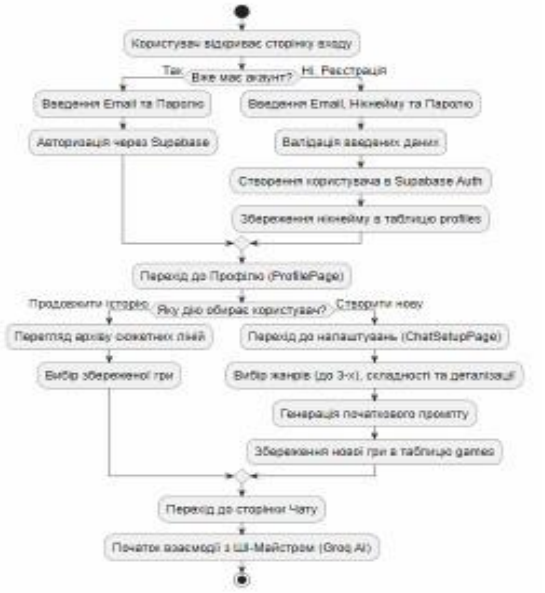


8

ДІАГРАМА ПОСЛІДОВНОСТІ ПРОЦЕСУ ОБРОБКИ ІГРОВОГО ХОДУ

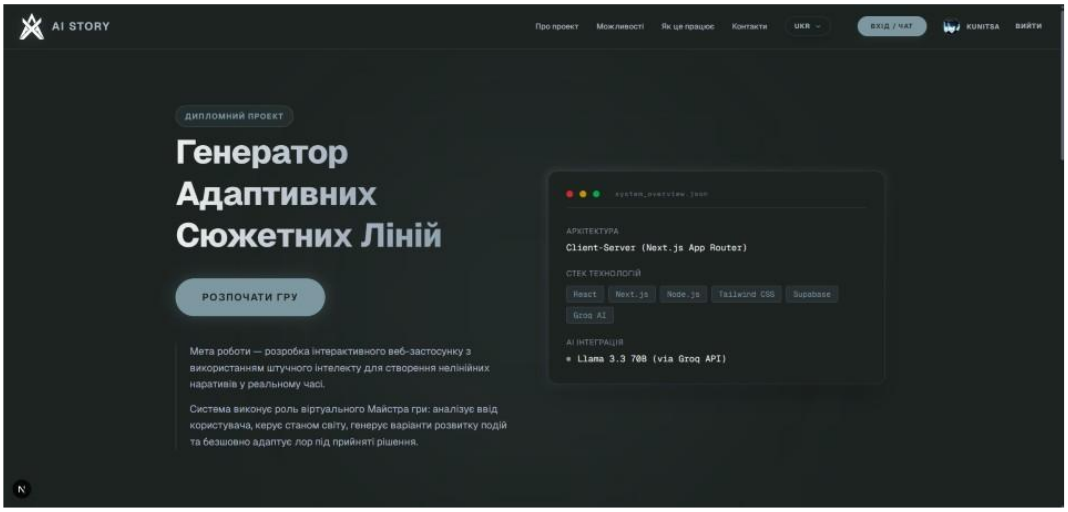
9

ДІАГРАМА ДІЯЛЬНОСТІ



10

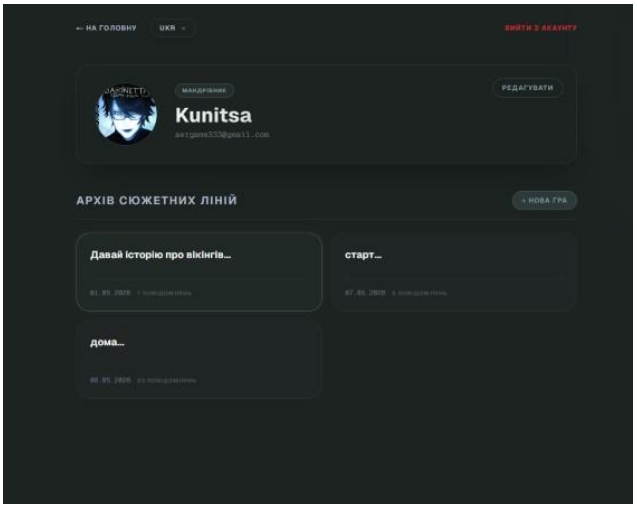
ЕКРАННІ ФОРМИ



Головна сторінка

9

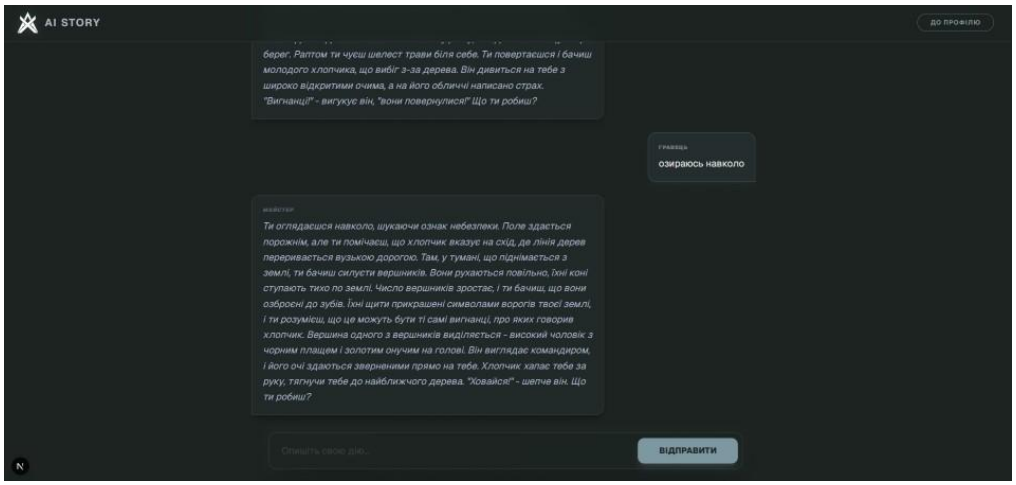
ЕКРАННІ ФОРМИ



Архів історій

10

ЕКРАННІ ФОРМИ



Чат з описом історії

11

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Мацела М.Р. Задонцев Ю.В. Визначення вимог до розробки веб-застосунку для генерації адаптивних сюжетних ліній з використанням штучного інтелекту. II Всеукраїнська науково-технічна конференція "Виклики та рішення в програмній інженерії", 26 листопада 2025 року, м. Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. С. 327-329.

2. Мацела М.Р. Задонцев Ю.В. Архітектурні патерни інтеграції великих мовних моделей (LLM) у клієнтські вебзастосунки. Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація», 1-3 травня 2026 року, м. Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. (подано до друку).

14

ВИСНОВКИ

1. Проведено аналіз предметної галузі, методів генерації динамічних сюжетів, процедурної генерації контенту та інтерактивних наративів, що виявив критичні обмеження статичних сценаріїв у сучасних ігрових системах.
2. Проаналізовано великі мовні моделі (LLM) та проведено порівняльний аналіз архітектур Llama 3.3, GPT-4 і Claude, за результатами якого обґрунтовано вибір Llama 3.3 як оптимального генеративного двигуна.
3. Проведено аналіз існуючих технологій та інструментів, на основі якого обрано технологічний стек із використанням інфраструктури Groq LPU та бази даних Supabase для швидкої обробки запитів.
4. Спроєктовано архітектуру системи та розроблено схеми взаємодії між клієнтською частиною (Frontend), Edge-функціями та AI-провайдером, що дозволило мінімізувати накладні витрати на передачу даних.
5. Розроблено та реалізовано інтелектуальну систему StoryTaller на базі фреймворку Next.js 15, яка забезпечує генерацію адаптивних наративів із підтримкою збереження контексту в PostgreSQL для гарантії цілісності сюжету.
6. Проведено перевірку працездатності інтегрованих програмних модулів та тестування кінцевого вигляду продукту. Результати підтверджують коректне виконання закладених функцій: успішну маршрутизацію запитів, цілісність генерації сюжету та безперебійну взаємодію клієнтської частини з AI-провайдером.

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```
"use client";
import { useState, useEffect } from "react";
import { useRouter } from "next/navigation";
import { supabase } from "@/utils/supabase";
import Link from "next/link";

export default function LoginPage() {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [username, setUsername] = useState(""); // Новий стан для нікнейму
  const [isLogin, setIsLogin] = useState(true);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  const router = useRouter();

  useEffect(() => {
    if (typeof window !== "undefined") {
      const urlParams = new URLSearchParams(window.location.search);
      if (urlParams.get("mode") === "register") {
        setIsLogin(false);
      }
    }
  }, []);

  // Функції валідації (на основі твоїх вимог)
  const isUsernameValid = (name) => {
    // Тільки букви, цифри, поодинокі дефіси. Не може починатися/закінчуватися дефісом. Довжина 3-20.
    const regex = /^[a-zA-Z0-9]+(-[a-zA-Z0-9]+)*$/;
    return regex.test(name) && name.length >= 3 && name.length <= 20;
  };

  const isPasswordValid = (pwd) => {
    // Або >= 15 символів, або >= 8 символів + цифра + мала літера
    const hasNumber = /\d/.test(pwd);
    const hasLowercase = /[a-z]/.test(pwd);
    return pwd.length >= 15 || (pwd.length >= 8 && hasNumber && hasLowercase);
  };

  const handleAuth = async (e) => {
    e.preventDefault();
    setLoading(true);
    setError(null);

    try {
      if (isLogin) {
        // --- ЛОГІКА ВХОДУ ---
        const { error: signInError } = await supabase.auth.signInWithPassword({ email, password });
        if (signInError) throw signInError;
        router.push("/profile");

      } else {
        // --- ЛОГІКА РЕЄСТРАЦІЇ ---

        // 1. Валідація на фронтенді
        if (!isUsernameValid(username)) {
          throw new Error("Нікнейм може містити лише літери, цифри та поодинокі дефіси (від 3 до 20 символів).");
        }
        if (!isPasswordValid(password)) {
          throw new Error("Пароль має містити мінімум 15 символів АБО 8 символів (включаючи цифру та малу літеру).");
        }

        // 2. Перевірка унікальності нікнейму в базі
        const { data: existingUser, error: checkError } = await supabase
          .from("profiles")
          .select("username")
          .ilike("username", username) // ilike робить перевірку нечутливою до регістру (Test = test)
          .maybeSingle();

        if (checkError) throw checkError;
        if (existingUser) throw new Error("Цей нікнейм вже зайнятий. Оберіть інший.");

        // 3. Створення користувача в Supabase Auth
        const { data: authData, error: signUpError } = await supabase.auth.signUp({ email, password });
        if (signUpError) throw signUpError;

        // 4. Збереження нікнейму в таблицю profiles
        if (authData.user) {
          const { error: profileError } = await supabase.from("profiles").insert([
            { id: authData.user.id, username: username }
          ]);
          if (profileError) throw profileError;
        }

        router.push("/profile");
      }
    } catch (err) {
      setError(err.message === "Invalid login credentials" ? "Невірний email або пароль" : err.message);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="min-h-screen bg-carbon text-platinum font-sans flex flex-col items-center justify-center relative selection:bg-steel selection:text-carbon px-4 py-12">
      <div className="absolute top-1/2 left-1/2 -translate-x-1/2 -translate-y-1/2 w-[600px] h-[600px] bg-slate/10 rounded-full blur-[120px] pointer-events-none"></div>

      <Link href="/" className="absolute top-8 left-8 text-powder hover:text-platinum transition-colors text-sm font-bold tracking-widest uppercase flex items-center gap-2">
        <span>← На головну</span>
      </Link>

      <div className="w-full max-w-md relative z-10">
        <div className="text-center mb-8">
          <div className="w-16 h-16 mx-auto mb-4">
            <img src="/logo.svg" alt="Logo" className="w-full h-full object-contain drop-shadow-[0_0_15px_rgba(125,152,161,0.4)]" />
          </div>
          <h1 className="text-3xl font-extrabold tracking-tight text-transparent bg-clip-text bg-gradient-to-r from-platinum to-steel">
            {isLogin ? "Повернення до гри" : "Новий мандрівник"}
          </h1>
        </div>

        <div className="bg-carbon/80 backdrop-blur-xl border border-slate/30 p-8 rounded-3xl shadow-2xl">
```

```
        <form onSubmit={handleAuth} className="space-y-5">

          {/* Поле Email */}
          <div>
            <label className="block text-xs font-bold text-
powder/70 uppercase tracking-widest mb-2">Електронна
пошта *</label>
            <input
              type="email"
              value={email}
              onChange={(e) => setEmail(e.target.value)}
              required
              className="w-full bg-slate/10 border border-slate/30
rounded-xl px-4 py-3 text-platinum placeholder:text-powder/30
focus:outline-none focus:border-steel/50 transition-colors"
              placeholder="mandrivnyk@axis.com"
            />
          </div>

          {/* Поле Нікнейму (Тільки для реєстрації) */}
          {!isLogin && (
            <div>
              <label className="block text-xs font-bold text-
powder/70 uppercase tracking-widest mb-2">Нікнейм
*</label>
              <input
                type="text"
                value={username}
                onChange={(e) => setUsername(e.target.value)}
                required
                className="w-full bg-slate/10 border border-
slate/30 rounded-xl px-4 py-3 text-platinum placeholder:text-
powder/30 focus:outline-none focus:border-steel/50 transition-
colors"
                placeholder="User"
              />
              <p className="text-[11px] text-powder/50 mt-2
leading-relaxed">
                Може містити лише літери, цифри та поодинокі
дефіси (не на початку/кінці).
              </p>
            </div>
          )}

          {/* Поле Пароля */}
          <div>
            <label className="block text-xs font-bold text-
powder/70 uppercase tracking-widest mb-2">Пароль *</label>
            <input
              type="password"
              value={password}
              onChange={(e) => setPassword(e.target.value)}
              required
              className="w-full bg-slate/10 border border-slate/30
rounded-xl px-4 py-3 text-platinum placeholder:text-powder/30
focus:outline-none focus:border-steel/50 transition-colors"
              placeholder="Введіть пароль"
            />
            {!isLogin && (
              <p className="text-[11px] text-powder/50 mt-2
leading-relaxed">
                Пароль має містити мінімум 15 символів АБО
мінімум 8 символів (включаючи цифру та малу літеру).
              </p>
            )}
          </div>

          {error && (
            <div className="p-3 bg-red-500/10 border border-red-
500/30 rounded-lg text-red-400 text-sm text-center">
              {error}
            </div>
          )}

          <button
            type="submit"
            disabled={loading}
            className="w-full py-4 bg-steel text-carbon font-black
rounded-xl hover:bg-platinum transition-all active:scale-95
shadow-lg shadow-steel/20 disabled:opacity-50 disabled:cursor-
not-allowed uppercase tracking-widest mt-2"
          >
            {loading ? "Обробка..." : (isLogin ? "Увійти" :
"Зареєструватися")}
          </button>
        </form>

        <div className="mt-6 text-center border-t border-
slate/20 pt-6">
          <p className="text-powder/60 text-sm">
            {isLogin ? "Ще не маєте акаунту?" : "Вже є
акаунт?"}{" "}
            <button
              type="button"
              onClick={() => {
                setIsLogin(!isLogin);
                setError(null);
                setUsername(""); // Очищаємо нікнейм при
перемиканні
              }}
              className="text-steel hover:text-platinum font-bold
transition-colors"
            >
              {isLogin ? "Створити зараз" : "Увійти"}
            </button>
          </p>
        </div>
      </div>
    </div>
  );
}
```

```
"use client";
import { useState, useEffect } from "react";
import Link from "next/link";
import { supabase } from "@/utils/supabase";
import { useParams, useRouter } from "next/navigation";

export default function ActiveChatPage() {
  const { id } = useParams();
  const router = useRouter();

  const [messages, setMessages] = useState([]);
  const [input, setInput] = useState("");
  const [isLoading, setIsLoading] = useState(true);

  // СУПЕР-НАДІЙНИЙ СКРОЛ В САМИЙ НИЗ
  const scrollToBottom = () => {
    setTimeout(() => {
      // Прокручуємо сам блок з повідомленнями до
максимуму
      const chatContainer = document.getElementById("chat-
scroll-container");
      if (chatContainer) {
        chatContainer.scrollTo({
          top: chatContainer.scrollHeight,
          behavior: "smooth",
        });
      }

      // Прокручуємо все вікно браузера в самий низ
      window.scrollTo({
        top: document.body.scrollHeight,
        behavior: "smooth",
      });
    }, 150);
  };
```

```
  // ОСЬ ЦЕЙ БЛОК ЗНИК МИНУЛОГО РАЗУ! Він слідкує за оновленнями і викликає скрол
  useEffect(() => {
    scrollToBottom();
  }, [messages, isLoading]);

  // Функція для витягування варіантів та очищення тексту
  const parseMessage = (text) => {
    if (!text) return { cleanText: "", options: [] };
    const optionRegex = /\[Варіант:\s*(.*?)\]/g;
    const options = [];
    let match;

    while ((match = optionRegex.exec(text)) !== null) {
      options.push(match[1].trim());
    }

    const cleanText = text.replace(optionRegex, "").trim();
    return { cleanText, options };
  };

  // Завантаження історії при відкритті
  useEffect(() => {
    const fetchGame = async () => {
      const { data: { session } } = await supabase.auth.getSession();
      if (!session) {
        router.push("/login");
        return;
      }

      const { data, error } = await supabase
        .from("games")
        .select("*")
        .eq("id", id)
        .single();

      if (error || !data) {
        router.push("/profile");
        return;
      }

      setMessages(data.history || []);
      setIsLoading(false);
    };

    fetchGame();
  }, [id, router]);

  // Універсальна функція відправки
  const handleSend = async (forcedText = null) => {
    const playerText = forcedText || input;
    if (!playerText.trim() || isLoading) return;

    setInput("");
    setIsLoading(true);

    const playerMsg = { id: Date.now(), role: "player", text: playerText };
    const newHistory = [...messages, playerMsg];
    setMessages(newHistory);

    const typingId = Date.now() + 1;
    setMessages((prev) => [...prev, { id: typingId, role: "master", text: "Майстер формує реальність..." }]);

    try {
      const res = await fetch("/api/chat", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ messages: newHistory }),
      });

      const data = await res.json();
      const aiResponseText = data.response || data.error;

      const finalHistory = [...newHistory, { id: typingId, role: "master", text: aiResponseText }];
      setMessages(finalHistory);

      let updateData = { history: finalHistory };
      if (messages.length === 1) {
        updateData.title = playerText.slice(0, 30) + "...";
      }

      await supabase.from("games").update(updateData).eq("id", id);
    } catch (error) {
      setMessages((prev) =>
        prev.map((msg) =>
          msg.id === typingId ? { ...msg, text: "Помилка зв'язку з ефіром." } : msg
        )
      );
    } finally {
      setIsLoading(false);
    }
  };

  // Визначаємо варіанти для останнього повідомлення
  const lastMessage = messages[messages.length - 1];
  const isMasterLast = lastMessage?.role === "master" || lastMessage?.role === "assistant";
  const { options: currentOptions } = isMasterLast ? parseMessage(lastMessage.text) : { options: [] };

  return (
    <div className="min-h-screen bg-carbon text-platinum font-sans flex flex-col selection:bg-steel selection:text-carbon">

      {/* ХЕДЕР */}
      <header className="px-8 py-4 flex items-center justify-between border-b border-slate/30 bg-carbon/90 backdrop-blur-md sticky top-0 z-40">
        <Link href="/" className="flex items-center gap-3 hover:opacity-80 transition-opacity">
          <div className="w-10 h-10 flex items-center justify-center shrink-0">
            <img src="/logo.svg" alt="Logo" className="w-full h-full object-contain drop-shadow-[0_0_10px_rgba(125,152,161,0.3)]" />
          </div>
          <span className="text-xl font-bold tracking-widest text-powder uppercase">AI Story</span>
        </Link>
        <Link href="/profile">
          <button className="px-6 py-2 border border-steel/50 text-steel font-bold rounded-full hover:bg-steel hover:text-carbon transition-all text-xs tracking-widest uppercase">
            До профілю
          </button>
        </Link>
      </header>

      {/* ВІКНО ЧАТУ */}
      <main className="flex-1 flex flex-col max-w-5xl w-full mx-auto p-4 md:p-8 overflow-hidden">

        {/* КОНТЕЙНЕР ЗІ СКРОЛОМ */}
        <div id="chat-scroll-container" className="flex-1 overflow-y-auto space-y-6 pr-2 pb-8 scrollbar-thin scrollbar-thumb-slate scrollbar-track-carbon">

          {/* Повідомлення */}
          {messages.map((msg) => {
            const { cleanText } = parseMessage(msg.text);
            return (
              <div
                key={msg.id}
```

```
            className={`flex ${msg.role === "player" ? "justify-
end" : "justify-start"} animate-in fade-in slide-in-from-bottom-2
duration-300`}
          >
            <div className={`max-w-[85%] md:max-w-[70%] p-
5 rounded-2xl border ${
              msg.role === "player"
              ? "bg-steel/10 border-steel/30 text-platinum rounded-
br-none shadow-lg shadow-steel/5"
              : "bg-slate/10 border-slate/30 text-powder rounded-
bl-none italic shadow-lg shadow-black/20"
            }`}>
              <div className="text-[10px] uppercase tracking-
widest mb-2 opacity-40 font-black">
                {msg.role === "player" ? "Гравець" : "Майстер"}
              </div>
              <div className="text-lg leading-relaxed
whitespace-pre-wrap">{cleanText}</div>
            </div>
          </div>
        );
      })}

      {/* БЛОК ВАРІАНТІВ (КНОПКИ) */}
      {isMasterLast && currentOptions.length > 0 &&
!isLoading && (
        <div className="w-full max-w-4xl mx-auto flex flex-
col gap-2 mt-6 animate-in fade-in slide-in-from-bottom-4
duration-500">
          {currentOptions.map((option, index) => (
            <button
              key={index}
              onClick={() => handleSend(option)}
              className="w-full text-left p-4 rounded-xl border
border-steel/20 bg-slate/5 hover:bg-steel/10 hover:border-
steel/60 transition-all group relative overflow-hidden backdrop-
blur-sm shadow-sm"
            >
              <div className="absolute left-0 top-0 bottom-0 w-1
bg-steel/20 group-hover:bg-steel transition-colors"></div>
              <div className="flex items-center gap-4 pl-2">
                <span className="text-steel/40 font-mono text-xs
group-hover:text-steel">0{index + 1}</span>
                <span className="text-powder group-hover:text-
platinum text-base">{option}</span>
              </div>
            </button>
          ))}
        </div>
      )}
    </div>

    {/* ПОЛЕ ВВОДУ (завжди знизу) */}
    <div className="relative group max-w-4xl mx-auto w-full
shrink-0">
      <div className="absolute -inset-1 bg-gradient-to-r from-
steel/30 to-slate/30 rounded-2xl blur opacity-20 group-focus-
within:opacity-60 transition-opacity duration-500"></div>

      <div className="relative flex items-center gap-4 bg-
carbon/80 backdrop-blur-xl border border-slate/30 p-2 rounded-
2xl shadow-2xl">
        <input
          type="text"
          value={input}
          onChange={(e) => setInput(e.target.value)}
          onKeyDown={(e) => e.key === "Enter" &&
handleSend()}
          placeholder={isLoading ? "Майстер формує
реальність..." : "Опишіть свою дію..."}
          disabled={isLoading}
          className="flex-1 bg-transparent border-none outline-
none px-4 py-3 text-platinum placeholder:text-powder/20 text-lg
disabled:opacity-50"
        />
        <button
          onClick={() => handleSend()}
          disabled={isLoading}
          className="px-10 py-3 bg-steel text-carbon font-black
rounded-xl hover:bg-platinum hover:scale-[1.02] transition-all
active:scale-95 shadow-lg shadow-steel/20 disabled:opacity-50"
        >
          ВІДПРАВИТИ
        </button>
      </div>
    </div>
  </main>
 </div>
 );
}
```