Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету(відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: «Моделювання адаптивного рівня складності та ефективності запам'ятовування у навчальних іграх»

Виконав студент 6 курсу, групи КН-61
спеціальності:
122 „Комп'ютерні науки"
(шифр і назва напрямку підготовки спеціальності)

Туровський Ілля Григорович
(прізвище, ім'я, по батькові)

Керівники: Сінкевич О.В., Шиманський В.М.
(прізвище, ініціали)

Рецензент: Часковський О.І.
(прізвище, ініціали)

Львів-2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"

ЗАТВЕРДЖУЮ:
Завідувачка кафедри КН
_____Борецька І.Б.
"10" грудня 2025 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Туровський Ілля Григорович
(прізвище, ім'я, по батькові)

1.Тема магістерської роботи: "Моделювання адаптивного рівня складності та ефективності запам'ятовування у навчальних іграх", затверджена наказом вищого навчального закладу від "29" квітня 2025 року, № С-288.
2.Термін подання студентом проекту(роботи) "10" грудня 2025 року.
3. Вихідні дані до проекту (роботи) Розробити веб-платформу «Memory+» - адаптивну навчальну гру типу «Memory», що дозволяє тренувати короткочасну пам'ять та увагу користувачів (насамперед дітей). Для розроблення програмного забезпечення використати мову програмування Java Script.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Стан проблемної області
Інформаційне забезпечення
Математичне забезпечення
Програмне забезпечення
Розроблення стартап-проєкту
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка матеріалу до доповіді.
6. Дата видачі завдання 1 травня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Етапи магістерської роботи	Термін виконання	Відмітка про виконання
1.	Отримання звітів продаж	1-5 травня	виконано
2.	Обрання алгоритмів рекомендації	6-12 травня	виконано
3.	Обрання мов програмування	13-18 травня	виконано
4.	Обрання моделі даних та математичних моделей	19-30 травня	виконано
5.	Проєктування системи	1-15 червня	виконано
6.	Розробка фронтенду	16 червня - 10 вересня	виконано
7.	Розробка бекенду	11 вересня - 15 жовтня	виконано
8.	Проведення тестування системи	16 жовтня - 15 листопада	виконано
9.	Оформлення пояснювальної записки	16 листопада - 10 грудня	виконано

Студент _____ Туровський І.Г.
(підпис) (прізвище та ініціали)

Керівники роботи _____ Сінкевич О.В.
(підпис) (прізвище та ініціали)

_____ Шиманський В.М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська робота містить 156 сторінок пояснювальної записки, 75 рисунків, 2 додатки, 29 джерел.

Основною метою роботи є розробка інтерактивної навчальної гри типу «Memory» з адаптивним моделюванням рівня складності та оцінюванням ефективності запам’ятовування користувача. У роботі реалізовано інтеграцію математичних методів, таких як рейтингова система Elo, ентропія Шеннона, експоненціальне згладжування та алгоритм інтервального повторення SuperMemo-2, що дозволяють автоматично коригувати складність гри залежно від результатів гравця. Розроблено веб-платформу, яка забезпечує взаємодію з ігровим процесом, відображення статистики та збереження прогресу користувача. У якості реалізації використано HTML, CSS, JavaScript (React), Node.js та базу даних MongoDB/PostgreSQL.

Ключові слова: адаптивна навчальна гра, Memory, когнітивні функції, рівень складності, Elo, ентропія Шеннона, SuperMemo-2, React, Node.js.

ABSTRACT

The thesis contains 156 pages of explanatory note, 75 figures, 2 appendices and 29 sources.

The main goal of the work is the development of an interactive “Memory” educational game with adaptive difficulty modeling and evaluation of memorization efficiency. The system integrates mathematical methods such as the Elo rating model, Shannon entropy, exponential smoothing and the SuperMemo-2 spaced repetition algorithm, which allow automatic adjustment of game difficulty based on user performance. A web platform has been developed to provide a user interface for gameplay interaction, progress tracking and performance visualization. The implementation uses HTML, CSS, JavaScript (React), Node.js and MongoDB/PostgreSQL.

Keywords: adaptive educational game, Memory, cognitive skills, difficulty modeling, Elo, Shannon entropy, SuperMemo-2, React, Node.js.

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити адаптивну навчальну гру типу “Memory”, яка моделює рівень складності та аналізує ефективність запам’ятовування користувача.

Система повинна враховувати різний рівень підготовки гравців, адаптуючи ігрове поле, час на виконання завдань та порядок повторень карток на основі математичних моделей.

У роботі повинні бути реалізовані:

– алгоритм рейтингової оцінки складності (Elo);

– аналіз ймовірності успіху гравця на основі ентропії Шеннона;

– експоненціальне згладжування для прогнозування подальшої продуктивності;

– алгоритм оптимального повторення SuperMemo-2 для побудови системи адаптивних повторень.

Розробити веб-застосунок, що надасть користувачу інтерфейс для проходження гри, перегляду своєї статистики та збереження прогресу. Забезпечити інтеграцію фронтенду (React) та бекенду (Node.js) з використанням бази даних MongoDB/PostgreSQL для зберігання інформації про користувачів та результати гри.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

AI (Artificial Intelligence) – штучний інтелект, галузь комп'ютерних наук, що займається створенням алгоритмів, здатних до навчання та прийняття рішень.

API (Application Programming Interface) – інтерфейс прикладного програмування, який забезпечує взаємодію між фронтендом та бекендом.

БД – база даних.

Elo – рейтинговий алгоритм, що визначає рівень майстерності гравця на основі його успішності.

UI (User Interface) – користувацький інтерфейс.

UX (User Experience) – досвід взаємодії користувача з системою.

FR (Frontend) – частина веб-застосунку, з якою безпосередньо взаємодіє користувач.

BE (Backend) – серверна частина системи, що відповідає за обробку даних і бізнес-логіку.

SM-2 (SuperMemo-2) – алгоритм інтервального повторення для оптимізації процесу запам'ятовування.

JS (JavaScript) – мова програмування для створення інтерактивних веб-додатків.

HTML (HyperText Markup Language) – мова розмітки веб-сторінок.

CSS (Cascading Style Sheets) – мова стилів для оформлення веб-інтерфейсів.

SPA (Single Page Application) – веб-застосунок, що працює на одній сторінці без повного перезавантаження.

CRUD (Create, Read, Update, Delete) – основні операції взаємодії з даними у базі.

DTO (Data Transfer Object) – об'єкт для передачі структурованих даних між компонентами системи.

JWT (JSON Web Token) – стандарт авторизації та автентифікації користувачів у веб-застосунках.

ВСТУП

З розвитком цифрових технологій та онлайн-освіти значно зростає кількість даних про процеси навчання та результати користувачів у навчальних іграх. Це створює потребу у створенні інтелектуальних систем, здатних аналізувати індивідуальні показники та підлаштовувати складність завдань під рівень користувача. Адаптивний підхід дозволяє підвищити ефективність запам’ятовування, розвиток уваги та мотивацію до регулярних тренувань.

Магістерська робота присвячена розробці веб-додатку на основі гри Memory, який реалізує динамічне налаштування рівня складності для користувачів різного віку та підготовки. Для дослідження були використані експериментальні дані про виконання завдань користувачами та результати попередньої бакалаврської роботи, що стали вихідною базою для формування адаптивних алгоритмів підбору завдань.

Важливим аспектом є створення системи, яка враховує історію ігрової активності користувача та застосовує алгоритми Elo, Shannon Entropy, Exponential Smoothing та SuperMemo-2 для оцінки прогресу та підбору оптимального рівня складності. Такий підхід дозволяє реалізувати персоналізоване навчання та підтримувати мотивацію користувачів завдяки гейміфікації.

Об’єктом дослідження є веб-додаток для тренування пам’яті та уваги, який регулює рівень складності завдань у грі Memory.

Метою роботи є розробка та впровадження адаптивної системи на веб-платформі (HTML/CSS/JS/React + Node.js + БД), що підвищує ефективність розвитку пам’яті та уваги користувачів через персоналізовані завдання у грі Memory.

Практичне значення полягає у створенні інтерактивного інструменту для тренування когнітивних навичок, що дозволяє автоматично підлаштовувати складність завдань під індивідуальні потреби користувача, підвищуючи ефективність навчання та рівень залученості.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Актуальність проблеми розвитку пам’яті та уваги через навчальні ігри

У сучасному інформаційному суспільстві формування та підтримання когнітивних функцій є важливою умовою успішної навчальної, професійної та соціальної діяльності. Поступове зростання обсягів інформації, що потребує оперативного опрацювання, підвищує вимоги до здатності людини швидко сприймати, утримувати й відтворювати дані. При цьому найбільше навантаження припадає на короткочасну та робочу пам’ять, а також на механізми уваги, які забезпечують фільтрацію та структуризацію інформаційних потоків. Зниження ефективності цих функцій призводить до погіршення результатів навчання, зменшення продуктивності праці та підвищення когнітивної втоми.

Когнітивні процеси належать до категорії тих характеристик нервової системи, які можуть бути цілеспрямовано розвинені за допомогою спеціальних вправ і методик. Нейропсихологічні дослідження підтверджують, що пластичність нервової системи зберігається впродовж усього життя, що дозволяє застосовувати тренувальні методики як для дітей, так і для дорослих, включаючи людей похилого віку. Це визначає актуальність створення доступних і науково обґрунтованих інструментів, здатних забезпечувати систематичний розвиток пам’яті та пов’язаних із нею когнітивних функцій.

Упродовж останніх років особливу увагу дослідників привертають навчальні ігри як засіб тренування когнітивних процесів. На відміну від традиційних форм навчання, які передбачають виконання повторюваних завдань, навчальні ігри містять ігрові механізми, що сприяють залученню користувача до процесу пізнання. Вони поєднують мотиваційні чинники з когнітивним навантаженням, що створює сприятливі умови для формування навичок. Додатковою перевагою таких ігор є здатність моделювати ситуації, в яких робоча пам’ять та увага функціонують у режимі наближеному до реальних умов.

Серед широкого спектра навчальних ігор помітне місце займають ігри типу Memory, у яких користувачеві необхідно запам’ятати розташування візуальних об’єктів та знаходити відповідні пари. Завдяки простоті та чіткості механізму така

форма діяльності дозволяє тренувати базові когнітивні процеси, зокрема візуально-просторову пам’ять, оперативну пам’ять, концентрацію уваги та здатність до порівняння образів. Memory використовується не лише як розважальна гра, а й як методичний інструмент у педагогічній, реабілітаційній та діагностичній практиці, оскільки дає змогу спостерігати закономірності закріплення та відтворення інформації.

Попри розповсюдженість і методичну цінність, класичні варіанти Memory не враховують індивідуальних особливостей користувача. Їхня статична структура призводить до того, що складність не змінюється відповідно до рівня навченості чи поточних можливостей людини. Це знижує ефективність тренування, оскільки виконання завдань, що є надто простими або надмірно складними, не сприяє формуванню нових когнітивних зв’язків. Така ситуація актуалізує питання побудови адаптивних систем[3], які здатні коригувати рівень складності відповідно до індивідуальних характеристик користувача.

Сучасні підходи до моделювання адаптивних навчальних систем ґрунтуються на використанні математичних моделей аналізу поведінки користувача. Застосування рейтингових систем, зокрема алгоритму Elo, дає змогу здійснювати кількісну оцінку рівня підготовки користувача та визначати рівень складності завдань, який є для нього оптимальним. Теоретико-інформаційні методи, зокрема обчислення ентропії, дозволяють аналізувати стабільність виконання завдань та визначати рівень когнітивної невизначеності. Алгоритми прогнозування, до яких належить експоненціальне згладжування, використовуються для оцінки тенденцій у зміні результатів. Моделі інтервального повторення, зокрема SuperMemo 2, забезпечують можливість формування структурованих схем закріплення матеріалу, що підвищує ймовірність довготривалого збереження інформації.

Інтеграція перелічених підходів дає можливість створити навчальну систему, яка виконує кілька ключових функцій:

1. когнітивного тренажера, що підтримує систематичне тренування пам’яті;
2. інструмента реабілітації, який може бути використаний для відновлення когнітивних функцій після перенесених захворювань чи вікових змін;
3. модуля розвитку пам’яті для користувачів різного віку;
4. системи довготривалого навчання, здатної формувати індивідуальну траєкторію тренування.

З огляду на зазначені особливості, актуальність дослідження полягає у необхідності створення та аналізу навчальної гри, яка поєднує механіку Memory з адаптивними математичними моделями. Така система здатна забезпечити науково обґрунтований підхід до розвитку когнітивних навичок, створюючи умови для формування гнучкої та персоналізованої траєкторії навчання. Впровадження адаптивних механізмів у структуру ігрових вправ дозволяє підвищити ефективність тренування пам’яті, зменшити ризик когнітивного перевантаження та забезпечити сталі результати незалежно від вікових характеристик користувачів..

1.2. Когнітивні процеси пам’яті та уваги

Пам’ять і увага належать до фундаментальних когнітивних процесів, які забезпечують можливість сприйняття, збереження, обробки та відтворення інформації. У структурі психічної діяльності[15] вони займають центральне місце, визначаючи здатність людини до навчання, адаптації, прийняття рішень та організації поведінки. Дослідження цих процесів є важливим для побудови ефективних навчальних систем, зокрема таких, що використовують ігрові механізми для розвитку когнітивних навичок. Вивчення особливостей пам’яті та уваги дозволяє сформувати обґрунтовані вимоги до структури навчальних ігор, а також визначити методи оптимізації складності завдань.

Пам’ять у сучасних когнітивних моделях розглядається як система взаємопов’язаних механізмів, що забезпечують кодування, збереження та відтворення інформації. Традиційно виділяють сенсорну, короткочасну та довготривалу пам’ять.

Сенсорна пам’ять відповідає за швидке фіксування зорових і слухових подразників і має дуже короткий час утримання інформації. Короткочасна пам’ять забезпечує зберігання інформації протягом декількох секунд і є критично важливою при виконанні завдань на миттєве запам’ятовування. Саме вона відіграє ключову роль у виконанні вправ типу Memory, де користувачеві необхідно запам’ятовувати розташування зображень у межах обмеженого часу. Довготривала пам’ять зберігає інформацію протягом тривалого періоду та формується на основі повторюваних стимулів і узагальнення попереднього досвіду.

Важливим компонентом пам’яті є робоча пам’ять, яка забезпечує утримання та обробку інформації одночасно. На відміну від короткочасної пам’яті, яка зберігає дані у пасивній формі, робоча пам’ять здійснює активні операції порівняння, оновлення та інтеграції. Робоча пам’ять пов’язана з контролем уваги та використовується під час виконання інтелектуальних завдань. Багато наукових моделей розглядають її як операційну систему мозку, здатну керувати інформаційними потоками[2] та забезпечувати взаємодію між різними когнітивними механізмами. Тренування робочої пам’яті є важливим напрямом сучасної когнітивної психології, оскільки її стан безпосередньо пов’язаний зі здатністю до навчання.

Увага, як окремий когнітивний процес, забезпечує вибірковість сприйняття та контроль над інформаційними потоками. У структурі уваги традиційно виділяють такі компоненти, як концентрація, стійкість, переключення та розподіл. Концентрація уваги визначає здатність людини зосереджуватися на конкретному об’єкті, і є ключовою під час виконання завдань, що вимагають точності та швидкої реакції. Стійкість уваги визначає здатність підтримувати зосередженість протягом певного часу, що особливо важливо під час тривалих сесій у навчальних іграх. Переключення уваги забезпечує можливість переходу між різними аспектами завдання, а розподіл уваги дає змогу виконувати декілька когнітивних операцій паралельно.

У контексті навчальних ігор пам’ять та увага функціонують у взаємозв’язку, утворюючи єдину систему когнітивної діяльності. Вправа типу Memory активує одночасно декілька складових: короткочасне збереження візуальних образів, оновлення робочої пам’яті при відкритті нових карток, концентрацію уваги на

конкретних елементах і переключення уваги між різними частинами ігрового поля. Це робить механіку гри природним засобом тренування взаємопов'язаних когнітивних процесів. Від того, наскільки ефективно працює кожен із цих компонентів, залежить результативність виконання завдання.

Важливою характеристикою пам'яті є її обмежений обсяг. Короткочасна пам'ять здатна утримувати лише обмежену кількість елементів, що підтверджено класичними експериментами у сфері когнітивної психології. Збільшення кількості візуальних об'єктів у завданні Memory призводить до пропорційного підвищення когнітивного навантаження, оскільки користувачеві потрібно утримувати у свідомості більше позицій і зв'язків. Це створює обґрунтовану потребу у регулюванні складності завдання, що дозволяє підтримувати оптимальний рівень навантаження та уникати перевантаження пам'яті.

Увага також підпорядковується закономірностям обмеження. Під час виконання вправи користувач змушений постійно перерозподіляти ресурси між сприйняттям нових зображень, утриманням у пам'яті вже відкритих карток та обробкою результатів попередніх дій. За високої складності завдання увага швидко виснажується, що призводить до помилок та зниження ефективності. Це є одним із ключових аргументів на користь адаптивних систем, у яких рівень когнітивного навантаження регулюється відповідно до продуктивності користувача.

У сучасних дослідженнях когнітивних процесів значна увага приділяється індивідуальним відмінностям у роботі пам'яті та уваги. Люди з різним віком, рівнем підготовки чи типом діяльності демонструють різну швидкість обробки інформації та різну стійкість до когнітивного навантаження. Це створює необхідність індивідуального підходу, коли ефективність тренування забезпечується лише тоді, коли завдання відповідає поточному когнітивному стану та не виходить за межі оптимальної складності. Саме ця закономірність є основою для побудови адаптивних навчальних систем.

У грі Memory когнітивні процеси можна аналізувати через динаміку виконуваних дій. Кількість помилок, час відкриття пари, повторюваність помилкових ходів та стабільність успішних операцій є маркерами, які відображають стан пам'яті

та уваги. Такий підхід дозволяє не лише фіксувати результати, а й оцінювати когнітивні зміни, оскільки динаміка поведінки користувача демонструє рівень сформованості навичок. Використання цих даних у математичних моделях створює основу для побудови адаптивної навчальної гри, яка може виконувати функцію когнітивного тренажера, інструмента реабілітації, модуля розвитку пам’яті та системи довготривалого навчання.

Таким чином, дослідження пам’яті та уваги як ключових компонентів когнітивної діяльності є важливим етапом у формуванні вимог до навчальних ігор. Аналіз цих процесів дає змогу обґрунтувати необхідність адаптації складності, визначити механізми формування навичок[10] та окреслити можливості інтеграції математичних моделей у структуру навчального процесу. У подальших підрозділах розглядаються сучасні підходи до побудови адаптивних систем, що дозволяють забезпечити персоналізовану взаємодію з користувачем у межах ігрових завдань..

1.3. Адаптивні системи навчання та їх застосування у цифрових тренажерах

Адаптивні системи навчання розглядаються як один із провідних напрямів розвитку сучасних інформаційних технологій у сфері освіти та когнітивного тренування. На відміну від традиційних навчальних платформ, адаптивні системи передбачають автоматичне регулювання складності, змісту та послідовності подачі матеріалу відповідно до індивідуальних характеристик користувача. Такий підхід ґрунтується на аналізі динаміки виконання завдань, що дозволяє моделювати індивідуальну траєкторію розвитку та підтримувати оптимальний рівень когнітивного навантаження.

В основі адаптивних систем лежить принцип зворотного зв’язку, відповідно до якого навчальний інструмент реагує на результативність користувача, змінюючи складність наступного завдання або структуру навчальної діяльності. Реалізація зворотного зв’язку потребує використання методів, які забезпечують об’єктивне вимірювання рівня навченості, обробку показників ефективності та прогнозування подальшої динаміки. У цифрових тренажерах ці завдання виконуються за допомогою

математичних моделей, що визначають параметри адаптації на основі емпіричних даних поведінки користувача.

У науковій літературі виділяють кілька основних підходів до побудови адаптивних навчальних систем. Перший підхід базується на рейтингових моделях, що використовуються для оцінювання рівня майстерності у змагальних середовищах. Одним із найбільш відомих рішень є система Elo, яка дає можливість описати взаємозв'язок між складністю завдання та ймовірністю успішного виконання. У межах когнітивних тренажерів такі рейтингові системи можуть використовуватися для регулювання складності: чим вищий рівень користувача, тим більш складними стають завдання. Логіка роботи алгоритму Elo добре відповідає структурі ігор типу Memory, де результативність залежить від послідовності дій, кількості помилок, часу виконання та стабільності поведінки. У реалізації прототипу навчальної гри такі принципи відображено у модулі eloSystem.js, який визначає зміну рейтингу відповідно до результатів окремих сесій.

```
64  export function getDifficultyFromElo(elo, gameType = 'classic') {
65
66    if (elo < 700) {
67      return {
68        level: 'easy',
69        cardCount: 8,
70        revealTime: gameType === 'speedRecall' ? 4000 : 2500,
71        description: 'Початковий рівень'
72      };
73    } else if (elo < 1000) {
74      return {
75        level: 'easy-medium',
76        cardCount: 10,
77        revealTime: gameType === 'speedRecall' ? 3000 : 2000,
78        description: 'Легко-середній'
79      };
80    } else if (elo < 1300) {
81      return {
82        level: 'medium',
83        cardCount: 12,
84        revealTime: gameType === 'speedRecall' ? 2000 : 1500,
85        description: 'Середній'
86      };
87    } else if (elo < 1600) {
88      return {
89        level: 'medium-hard',
90        cardCount: 16,
91        revealTime: gameType === 'speedRecall' ? 1500 : 1200,
92        description: 'Середньо-складний'
93      };
94    } else {
95      return {
96        level: 'hard',
97        cardCount: 20,
98        revealTime: gameType === 'speedRecall' ? 1000 : 800,
99        description: 'Складний'
100     };
101   }
102 }
```

Рисунок 1.1 – Алгоритм Elo

Другий підхід ґрунтується на оцінюванні інформаційної складності завдання за допомогою методів теорії інформації. Ентропія, як міра невизначеності та варіативності результатів, дозволяє аналізувати, наскільки стабільно користувач виконує завдання і якою мірою гра створює когнітивне навантаження. У цифрових тренажерах ентропійний аналіз застосовується для визначення складних для користувача елементів, контролю динаміки запам'ятовування та виявлення систематичних помилок. У реалізації прототипу функції такого аналізу покладено на модуль microAnalysis.js, який формує оцінку стабільності виконання дій на основі історії спроб і успішності окремих карткових пар.

```
40  export function updatePairStats(pairStats, wasSuccessful) {
41    const updated = { ...pairStats };
42
43
44    updated.attempts += 1;
45    if (wasSuccessful) {
46      updated.success += 1;
47    }
48
49
50    updated.probability = updated.attempts > 0
51      ? updated.success / updated.attempts
52      : 0.5;
53
54
55    const S_t = wasSuccessful ? 1 : 0;
56    updated.prediction = ALPHA * S_t + (1 - ALPHA) * updated.prediction;
57
58
59    if (updateSM2Stats) {
60      const updatedWithSM2 = updateSM2Stats(updated, wasSuccessful);
61      updated.sm2 = updatedWithSM2.sm2;
62    } else {
63      /
64      if (!updated.sm2) {
65        updated.sm2 = {
66          eFactor: 2.5,
67          repetition: 0,
68          interval: 1,
69          nextReviewDate: null,
70          lastReviewDate: new Date().toISOString(),
71          totalReviews: 0,
72        };
73      }
74    }
75
76    return updated;
77  }
```

Рисунок 1.2 – Алгоритм ентропійного аналізу

Третій підхід орієнтований на прогнозування подальшої успішності користувача та визначення траєкторії розвитку навички. Для цього застосовуються статистичні моделі часових рядів і методи згладжування. Експоненціальне згладжування використовується для аналізу тенденцій у результатах користувача, враховуючи щойно отримані дані з більшою вагою, а минулі - з меншою. У контексті когнітивного тренажера такий підхід дозволяє визначити, чи покращується результативність користувача з часом, чи навпаки, спостерігається спадання ефективності. В ігровому прототипі прогнозування результатів реалізовано через обробку історії ігрових подій, які у подальшому можуть бути предметом окремого математичного моделювання в системі адаптації.

```
  const eloFactor = Math.min(1, (eloRating - 1000) / 1000);
  const weakPairsRatio = 0.2 + eloFactor * 0.2;
  const weakPairsCount = Math.max(1, Math.floor(targetPairCount * weakPairsRatio));
  const strongPairsCount = Math.max(1, Math.floor(targetPairCount * 0.4));

  const weakPairs = getWeakPairs(pairsStats);
  const strongPairs = getStrongPairs(pairsStats);

  const selectedPairs = [];
  const usedSymbols = new Set();

  for (let i = 0; i < Math.min(weakPairsCount, weakPairs.length); i++) {
    const symbol = weakPairs[i].pairId;
    if (!usedSymbols.has(symbol)) {
      selectedPairs.push(symbol);
      usedSymbols.add(symbol);
    }
  }

  for (let i = 0; i < Math.min(strongPairsCount, strongPairs.length); i++) {
    const symbol = strongPairs[i].pairId;
    if (!usedSymbols.has(symbol)) {
      selectedPairs.push(symbol);
      usedSymbols.add(symbol);
    }
  }

  const availableSymbols = allSymbols.filter(s => !usedSymbols.has(s));
  const remainingCount = targetPairCount - selectedPairs.length;

  for (let i = 0; i < Math.min(remainingCount, availableSymbols.length); i++) {
    selectedPairs.push(availableSymbols[i]);
  }

  while (selectedPairs.length < targetPairCount && allSymbols.length > 0) {
    const randomSymbol = allSymbols[Math.floor(Math.random() * allSymbols.length)];
    if (!selectedPairs.includes(randomSymbol)) {
      selectedPairs.push(randomSymbol);
    } else if (selectedPairs.length >= allSymbols.length) {
      break;
    }
  }

  return selectedPairs.slice(0, targetPairCount);
}
```

Рисунок 1.3 – Алгоритм експоненціального згладжування

Четвертий підхід пов’язаний з оптимізацією процесу повторення інформаційних елементів. Моделі інтервального повторення спрямовані на формування довготривалої пам’яті та ґрунтуються на закономірності, що інформація краще закріплюється за умови повторення з оптимальними інтервалами. Алгоритм SuperMemo 2 є одним із найбільш відомих рішень у цій сфері. Його застосування у когнітивних тренажерах дозволяє визначати, які елементи потребують повторення раніше, а які можна подавати із більшим часовим інтервалом. У розробленому прототипі ці правила реалізує модуль spacedRepetition.js, який забезпечує оновлення параметрів карткових пар залежно від стабільності їхнього запам’ятовування.

```
export function probabilityToQuality(probability, recentAttempts = null) {

  if (recentAttempts && recentAttempts.length > 0) {
    const recentSuccess = recentAttempts.filter(a => a.success).length;
    const recentRate = recentSuccess / recentAttempts.length;

    const combinedProb = (probability * 0.6 + recentRate * 0.4);

    if (combinedProb >= 0.9) return 5;
    if (combinedProb >= 0.75) return 4;
    if (combinedProb >= 0.6) return 3;
    if (combinedProb >= 0.4) return 2;
    if (combinedProb >= 0.2) return 1;
    return 0;
  }

  if (probability >= 0.9) return 5;
  if (probability >= 0.75) return 4;
  if (probability >= 0.6) return 3;
  if (probability >= 0.4) return 2;
  if (probability >= 0.2) return 1;
  return 0;
}


export function updateEFactor(currentEFactor, quality) {

  const newEFactor = currentEFactor + (0.1 - (5 - quality) * (0.08 + (5 - quality) * 0.02));

  return Math.max(MIN_E_FACTOR, newEFactor);
}

export function calculateInterval(repetition, eFactor) {
  if (repetition === 0) {
    return 1;
  } else if (repetition === 1) {
    return 6;
  } else {

    const baseInterval = 6;
    return Math.round(baseInterval * Math.pow(eFactor, repetition - 1));
  }
}
```

Рисунок 1.4 – Алгоритм SuperMemo 2

Адаптивні навчальні системи поєднують наведені підходи для формування комплексної моделі навчальної взаємодії. Така модель ґрунтується на зборі й аналізі даних, оновленні параметрів складності та прогнозуванні результатів. У контексті навчальної гри типу Memory інтеграція цих підходів дозволяє формувати індивідуальну траєкторію тренування, у якій складність гри, послідовність відкриття карток та інтервали повторення визначаються відповідно до когнітивної динаміки користувача. Це створює умови для поєднання навчального та ігрового змісту, забезпечуючи стабільну мотивацію та підвищуючи ефективність тренування пам’яті.

Сучасні цифрові рішення демонструють високу ефективність адаптивних підходів при формуванні когнітивних навичок. Їх використання дозволяє навчальним системам виконувати одночасно кілька функцій: функцію когнітивного тренажера, що забезпечує тренування пам’яті у контрольованому середовищі; функцію інструмента реабілітації, який може бути використаний для відновлення пам’яті та уваги після перенесених захворювань; функцію модуля розвитку пам’яті для користувачів різного віку; а також функцію системи довготривалого навчання, що забезпечує безперервний розвиток когнітивних якостей.

Таким чином, адаптивні системи навчання виступають важливим елементом сучасної цифрової освіти та тренування когнітивних навичок. Вони дозволяють поєднати алгоритмічні моделі та дані про поведінку користувача для побудови ефективної навчальної взаємодії. У рамках цієї роботи адаптивні підходи використовуються для створення навчальної гри, яка забезпечує автоматичне регулювання складності та оптимізацію процесів запам’ятовування.

1.4. Аналіз сучасних платформ когнітивного тренування та їх відповідність адаптивним підходам

Розвиток цифрових технологій спричинив появу значної кількості онлайн-платформ, спрямованих на тренування пам’яті, уваги, логічного мислення та інших когнітивних функцій. Ринок таких рішень формується під впливом інтересу користувачів до індивідуального розвитку пізнавальних здібностей, а також потреби професійних і освітніх установ у доступних інструментах для стимулювання когнітивної активності. Платформи, що набули найбільшого поширення, містять

набір вправ різних типів і ступенів складності, які дозволяють здійснювати регулярне тренування робочої пам’яті, швидкості мислення та когнітивної гнучкості.

Однією з найвідоміших платформ є Lumosity, яка використовує комплекс вправ для розвитку різних аспектів когнітивної діяльності. Система містить ігри, що тренують увагу, швидкість реакції, робочу пам’ять і гнучкість мислення. Основою платформи є персоналізований підбір вправ, який здійснюється відповідно до результатів користувача. Однак рівень адаптивності в Lumosity здебільшого ґрунтується на загальних показниках продуктивності, без детального аналізу структури помилок чи індивідуальних когнітивних характеристик. Також система не використовує математичних моделей глибокої адаптації, таких як інтервальні повторення чи ентропійний аналіз.

Платформа NeuroNation орієнтована на комплексний підхід до когнітивного тренування та містить вправи для розвитку пам’яті, уваги, мовлення й логічного мислення. На відміну від Lumosity, ця система частково враховує індивідуальні особливості користувачів шляхом коригування параметрів складності. Проте реалізація адаптації залишається дискретною та залежить від загальних показників продуктивності, що не дозволяє формувати чітку траєкторію розвитку навички. Відсутність математичних моделей, які оцінюють інформаційну складність або здійснюють прогнозування результатів, обмежує точність системи.

Іншим прикладом є Cognifit, який використовує інструменти оцінки когнітивного профілю користувача. Платформа формує персоналізовану програму тренувань на основі тестування, що включає аналіз різних когнітивних доменів. Попри розвинену діагностичну складову, Cognifit не забезпечує адаптації на рівні окремих елементів завдання. Параметри складності змінюються відповідно до узагальнених показників, а не до конкретних помилок, темпу навчання або стабільності результатів. Таким чином, система забезпечує високу початкову персоналізацію, але не реалізує повноцінний механізм індивідуального навчання в динаміці.

Багато сучасних платформ також включають тренажери типу Memory, проте їх реалізація, як правило, є статичною. Картки представлені у фіксованій кількості, а

порядок подачі завдань не змінюється залежно від результатів користувача. Навіть у випадках, коли система містить декілька рівнів складності, перехід між ними відбувається не автоматично, а за ініціативою користувача. У таких умовах когнітивне навантаження може бути або недостатнім, або надмірним, що знижує ефективність тренування.

Розглянуті системи мають істотну спільну особливість: вони забезпечують широкий набір вправ, але не реалізують глибокої адаптації, яка враховує всі аспекти поведінки користувача. У більшості випадків використовується узагальнена модель зміни складності, яка не реагує на індивідуальну динаміку розвитку навички. Це обмежує навчальний ефект, оскільки відсутність точного регулювання складності не дозволяє підтримувати оптимальний рівень когнітивного навантаження.

Потреба в інструментах адаптивного тренування підтверджується науковими дослідженнями, які демонструють, що ефективність когнітивного розвитку значною мірою залежить від ступеня збалансованості складності завдань. Надмірне спрощення призводить до відсутності прогресу, тоді як надмірне ускладнення викликає зростання кількості помилок і зниження мотивації. Саме тому сучасні тенденції розвитку навчальних ігор орієнтовані на використання алгоритмів, що забезпечують автоматичний контроль складності на основі аналізу поведінки користувача.

У цьому контексті навчальні системи, що використовують механіку гри Memory, є перспективними для впровадження адаптивних моделей. Механіка гри передбачає наявність численних параметрів, які можуть бути відображені у формальних моделях адаптації: кількість карток, час виконання завдань, послідовність відкриття карткових пар, стабільність запам’ятовування візуальних образів. Дані параметри можуть бути використані для аналізу когнітивних характеристик користувача й формування індивідуальної траєкторії розвитку навички.

На відміну від розглянутих платформ, адаптивна система, побудована на основі гри Memory, може застосовувати комплекс сучасних математичних моделей, що забезпечують точний аналіз поведінки користувача. Використання рейтингової моделі Elo дає змогу визначати рівень складності через зіставлення продуктивності користувача з умовним рівнем гри. Моделі ентропійного аналізу дозволяють оцінити

стабільність запам’ятовування карткових пар. Експоненціальне згладжування використовується для прогнозування успішності, а інтервальні повторення забезпечують формування довготривалої пам’яті.

Таким чином, аналіз сучасних платформ когнітивного тренування свідчить про наявність значних можливостей для удосконалення навчальних ігор за рахунок інтеграції адаптивних алгоритмів. Враховуючи обмеження розглянутих систем, доцільним є створення цифрового тренажера з глибокою адаптивністю, здатного одночасно виконувати функції когнітивного тренажера, інструмента реабілітації, модуля розвитку пам’яті та системи довготривалого навчання. Застосування математичних підходів, реалізованих у розробленому прототипі навчальної гри, дозволяє розширити можливості традиційних вправ типу Memory і забезпечити високий рівень персоналізації навчального процесу..

1.5. Аналіз попередньої бакалаврської роботи та напрями удосконалення

Попередня бакалаврська робота була присвячена створенню базової версії навчальної гри типу Memory, яка реалізовувала основну структуру ігрового процесу та функції збору статистики щодо дій користувача. Розроблена система дозволяла здійснювати елементарний аналіз поведінки користувача та оцінювати результативність виконання окремих ігрових завдань. У межах поставлених тоді завдань така система забезпечувала можливість початкового тренування короткочасної пам’яті та базових когнітивних процесів, проте її функціональні можливості були суттєво обмежені. Це зумовило необхідність подальшої розробки й поглиблення методологічної основи проєкту вже на рівні магістерської кваліфікаційної роботи.

Структура бакалаврського проєкту передбачала використання фіксованої кількості карток у межах одного рівня складності, а також незмінного алгоритму взаємодії з гравцем. Попри певну варіативність візуальних елементів та загальної послідовності дій, система не враховувала індивідуальних характеристик користувача та не забезпечувала автоматичного регулювання складності відповідно до рівня підготовки. У результаті, користувач із високою результативністю не отримував належного когнітивного навантаження, тоді як користувач із нижчою продуктивністю

міг зіштовхнутися із завданням, складність якого перевищувала його поточні можливості. За відсутності адаптивного механізму гра виконувала радше роль статичного тренажера, який не здатний підтримувати сталу траєкторію розвитку.

Іншою важливою особливістю попередньої реалізації було те, що система не включала інструментів аналізу історичних даних. Бакалаврська робота зосереджувалася переважно на фіксації кількості спроб, часу виконання та загальних показників ефективності. Такі показники мають обмежене значення для оцінювання когнітивного прогресу, оскільки вони не дають змоги визначити стабільність виконання завдань, складність окремих елементів і тенденції зміни результатів. Відсутність механізмів прогнозування, оцінки ентропії та аналізу часових послідовностей означала, що система не могла визначати потенційні труднощі користувача або адаптуватися до них.

З точки зору архітектури, попередня версія гри базувалася виключно на клієнтському збереженні даних. Вся інформація щодо статистики виконання задач зберігалася локально, що не дозволяло формувати агреговані дані, здійснювати порівняння користувачів або забезпечувати стабільне збереження прогресу у випадку зміни пристрою. Таке рішення було прийнятним у контексті бакалаврського рівня, проте воно обмежувало перспективи розширення системи та впровадження функцій, характерних для сучасних освітніх платформ.

Отже, аналіз попередньої роботи дозволяє виділити три основні обмеження, які стали підставою для подальшого удосконалення на магістерському рівні:

Перше обмеження - відсутність адаптивності. Для ефективного тренування пам’яті та уваги необхідно, щоб рівень складності змінювався відповідно до продуктивності користувача. Реалізація механізму адаптації[9] потребує математичних моделей, зокрема рейтингових та статистичних, що дозволяють формувати індивідуальну траєкторію навчання.

Друге обмеження - недостатність аналітичних можливостей. Для оцінки динаміки когнітивного розвитку необхідно враховувати не лише кількісні показники виконання завдань, а й якісні, які характеризують стабільність результатів, варіативність поведінки та потенційні проблеми із запам’ятовуванням певних

елементів. Це вимагає впровадження методів обчислення ентропії, експоненціального згладжування та інтервальних повторень.

Третє обмеження - архітектурна ізольованість. Відсутність централізованої обробки даних унеможливлювала розширення платформи за рахунок нових функцій, таких як система реєстрації, авторизації, лідерборди, порівняння користувачів, збереження статистики на сервері та багато інших елементів, які формують сучасні навчальні системи.

Магістерська кваліфікаційна робота покликана усунути зазначені обмеження та сформувати комплексну систему тренування когнітивних навичок. На відміну від попередньої версії, у межах представленої роботи створюється адаптивна платформа, заснована на використанні сучасних математичних моделей. Алгоритм Elo дозволяє кількісно оцінювати рівень складності відносно поточного рівня користувача. Моделі інформаційного аналізу та прогнозування забезпечують оцінку стабільності виконання завдань, а алгоритм SuperMemo 2 - можливість оптимізації процесу повторення інформаційних елементів. Усі ці підходи інтегруються у структуру навчальної гри та реалізуються у вигляді модулів, які забезпечують формування адаптивної траєкторії тренування.

Крім того, у межах магістерської роботи вдосконалено програмну архітектуру системи. Передбачено перехід до клієнт-серверної моделі[11], що дозволить зберігати статистику користувача у централізованій базі даних, забезпечувати доступ до результатів з різних пристроїв, реалізовувати персональні облікові записи та порівнювати ефективність різних користувачів у межах системи рейтингу. Це не лише сприяє підвищенню функціональності, але й створює умови для подальшого використання платформи у дослідницьких та освітніх цілях.

Таким чином, результати аналізу бакалаврської роботи демонструють потребу у суттєвому розширенні функціональних можливостей, математичного апарату та архітектури. Магістерська робота є логічним продовженням попереднього етапу та спрямована на створення комплексної адаптивної системи, здатної виконувати функції когнітивного тренажера, інструмента реабілітації, модуля розвитку пам’яті[8] та системи довготривалого навчання. Це формує методологічне підґрунтя для

подальших розділів, присвячених побудові інформаційного та математичного забезпечення системи.

1.6. Висновки до розділу 1

У межах першого розділу було здійснено аналіз проблемної області, що охоплює когнітивні процеси, сучасні навчальні ігри та наявні підходи до побудови адаптивних систем. Встановлено, що зростання інформаційного навантаження на користувачів різного віку зумовлює потребу у створенні інструментів для розвитку пам’яті та уваги, які забезпечують систематичне тренування когнітивних функцій. Навчальні ігри, зокрема ігри типу Memory, є ефективним засобом тренування короткочасної та робочої пам’яті, однак традиційні реалізації характеризуються статичною складністю і не враховують індивідуальні особливості користувача.

Розглянуті сучасні платформи когнітивного тренування засвідчують, що більшість таких систем використовують обмежені механізми адаптації, які не забезпечують повноцінного аналізу динаміки виконання завдань. Виявлено, що рейтингові моделі, методи інформаційного аналізу, алгоритми прогнозування та інтервального повторення дозволяють формувати персоналізовану траєкторію навчання, що робить їх перспективними для інтеграції у навчальні ігри.

Аналіз попередньої бакалаврської роботи показав, що створена раніше версія гри Memory була функціонально обмеженою, не містила інструментів адаптивності та не включала розширених засобів аналізу поведінки користувача. Відсутність математичного апарату, прогнозування та динамічного регулювання складності обмежувала її навчальний потенціал.

Таким чином, результати аналізу підтверджують доцільність створення адаптивної системи навчального призначення, заснованої на механіці гри Memory та доповненої сучасними математичними моделями.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Постановка завдань інформаційного забезпечення навчальної гри Memory+

Інформаційне забезпечення є ключовим компонентом побудови адаптивних навчальних систем, оскільки визначає структуру, зміст і організацію даних, необхідних для функціонування гри, аналізу поведінки користувача та формування індивідуальної траєкторії навчання. У контексті цифрового тренажера Memory+ інформаційне забезпечення охоплює всі види інформації, які використовуються системою: починаючи від ігрових елементів і характеристик користувача, закінчуючи статистикою виконання завдань, когнітивними показниками та даними для адаптивних алгоритмів. Визначення структури таких даних є базовим етапом для подальшого математичного моделювання та реалізації програмної архітектури системи.

Навчальні ігри, побудовані за принципом адаптивності, потребують ретельної організації інформаційних потоків, оскільки вся логіка персоналізованої взаємодії ґрунтується на даних, отриманих під час ігрового процесу. Саме інформаційне забезпечення створює інфологічну основу для роботи алгоритмів аналізу, прогнозування та регулювання складності. Це включає збирання даних про користувача, структуру ігрових об’єктів, фіксацію подій, облік помилок та визначення когнітивних параметрів, які впливають на роботу системи адаптації. Таким чином, інформаційне забезпечення виступає фундаментальною складовою, що визначає можливості системи щодо навчального впливу та ефективності тренування.

Постановка завдань інформаційного забезпечення передбачає визначення того, які саме дані необхідно збирати, зберігати та опрацьовувати для досягнення цілей навчальної гри. У системі Memory+ до основних завдань інформаційного забезпечення належать: формування інформаційної моделі користувача, опис структур ігрового середовища, побудова інформаційних потоків між компонентами гри, визначення параметрів статистики, яка використовується для адаптації рівня складності, а також організація зберігання інформації в архітектурі клієнт-серверної взаємодії.

Інформаційна модель користувача повинна забезпечувати фіксацію всіх параметрів, що характеризують індивідуальну траєкторію тренування. Це включає особистий рейтинг, історію ігрових сесій, параметри успішності, дані про стабільність виконання дій, рівні помилок та показники когнітивної динаміки. Така модель є необхідною для роботи алгоритму Elo, який визначає відповідний рівень складності, а також для аналізу в межах ентропійних оцінок[3] та моделей експоненціального згладжування[21]. У подальших підрозділах буде описано структуру цих даних та їх взаємозв'язок із математичним забезпеченням системи.

Другим важливим завданням є опис структури ігрового середовища, що включає картки, пари карток, рівні, часові обмеження, списки подій та інші елементи, які створюють інформаційний зміст гри. Ігрове середовище Memory+ має багаторівневу структуру, де кожна картка та кожна пара мають власні інформаційні параметри, що оновлюються залежно від дій користувача. Це дозволяє модулю microAnalysis формувати аналітичні показники, які в подальшому слугують базою для адаптивної зміни складності та оптимізації повторення завдань.

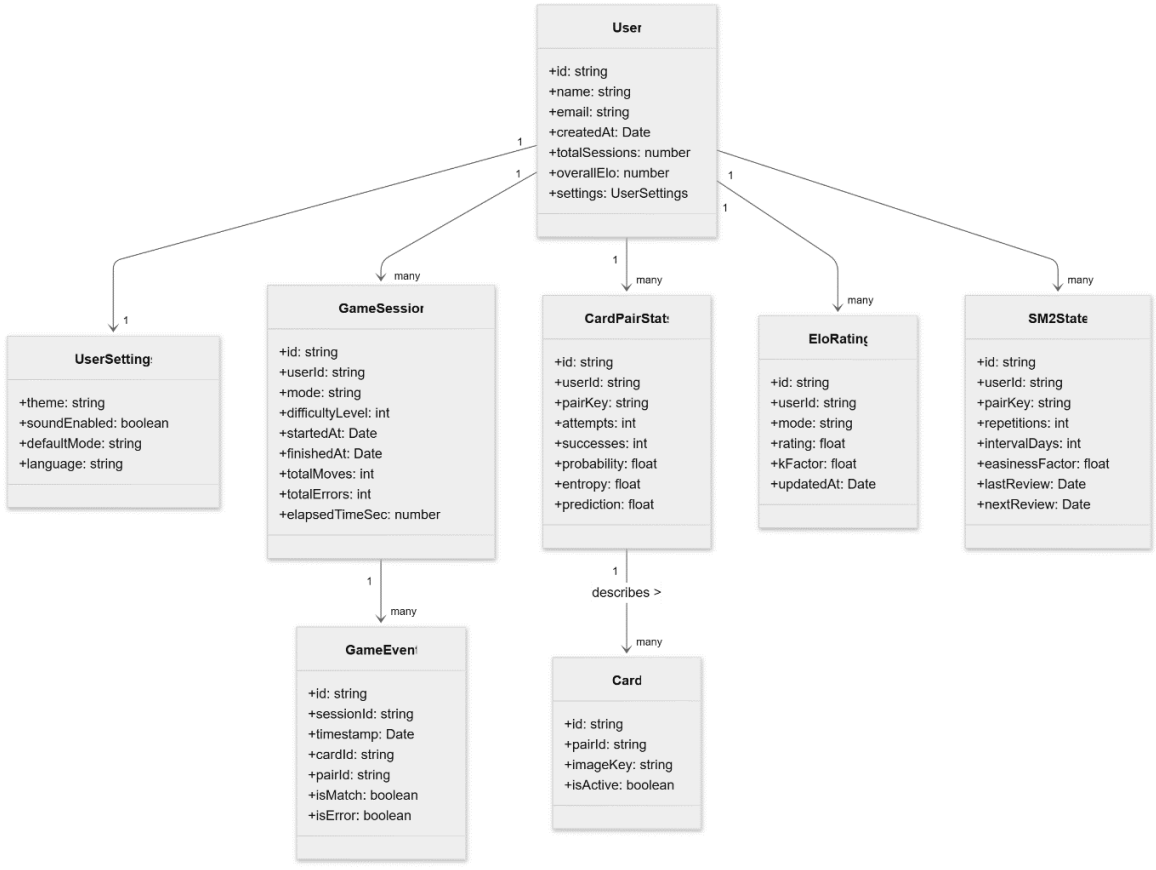


Рисунок 2.1 – Інформаційна модель ігрового середовища Memory+

Третє завдання полягає у формуванні інформаційних потоків між компонентами системи. Оскільки Memory+ поєднує ігрову механіку, систему адаптації та майбутній серверний модуль, інформаційні потоки включають обмін даними між інтерфейсом користувача, модулем статистики, адаптивними алгоритмами та системою зберігання. Ці потоки мають бути структурованими таким чином, щоб забезпечувати своєчасне оновлення параметрів, точну взаємодію з алгоритмами адаптації та коректне зберігання інформації у базі даних у серверній частині.

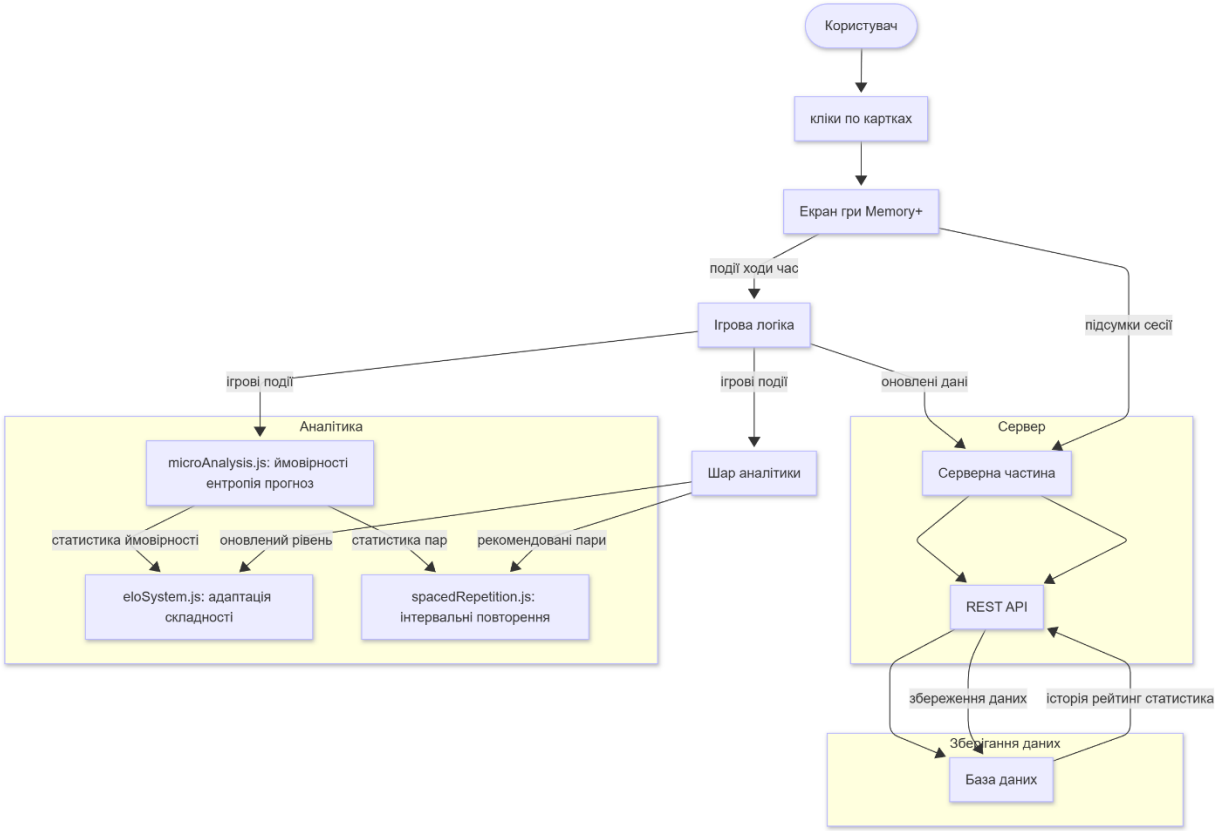


Рисунок 2.2 – Інформаційні потоки в системі Memory+

Ще одним аспектом є визначення параметрів статистики, яка використовується для адаптації рівня складності. Кількість помилкових ходів, час відкриття пар, стабільність запам’ятовування та інші показники формують інформаційну основу для математичних моделей. У файлі microAnalysis.js реалізовано обробку даних про кожну дію користувача, на основі яких обчислюється ентропія[21], ймовірність успішності та прогнозовані значення продуктивності. Такі дані становлять ключову частину інформаційного забезпечення, оскільки саме вони є основою адаптивної поведінки системи.

```
export function calculateEntropy(pairsStats) {
  if (!pairsStats || Object.keys(pairsStats).length === 0) {
    return 0;
  }

  let entropy = 0;
  const pairs = Object.values(pairsStats);

  for (const pair of pairs) {
    const p = pair.probability;

    if (p > 0 && p < 1) {
      entropy -= p * Math.log2(p);
    }
  }

  return entropy;
}


export function normalizeEntropy(entropy, pairCount) {
  if (pairCount <= 1) return 0;

  const maxEntropy = Math.log2(pairCount);
  return maxEntropy > 0 ? Math.min(1, entropy / maxEntropy) : 0;
}

export function calculateAverageSuccess(pairsStats) {
  if (!pairsStats || Object.keys(pairsStats).length === 0) {
    return 0.5;
  }

  const pairs = Object.values(pairsStats);
  const totalProbability = pairs.reduce((sum, pair) => sum + pair.probability, 0);

  return pairs.length > 0 ? totalProbability / pairs.length : 0.5;
}

export function calculateExpectedScore(pairsStats, pairCount) {
  const entropy = calculateEntropy(pairsStats);
  const normalizedEntropy = normalizeEntropy(entropy, pairCount);
  const averageSuccess = calculateAverageSuccess(pairsStats);

  const expectedScore = 1 - normalizedEntropy * (1 - averageSuccess);

  return Math.max(0, Math.min(1, expectedScore));
}
```

Рисунок 2.3 – Фрагмент реалізації статистичного аналізу дій користувача в модулі microAnalysis.js

Важливою складовою інформаційного забезпечення є також організація зберігання інформації. У поточній версії системи використовується локальне зберігання на клієнті, однак у повноцінній реалізації передбачено використання серверної бази даних, яка міститиме інформацію про користувача, статистику сесій, результати адаптивних алгоритмів та інші параметри. Це дозволить забезпечити

збереження прогресу, порівняння результатів, формування лідербордів та зручний доступ до даних з різних пристроїв.

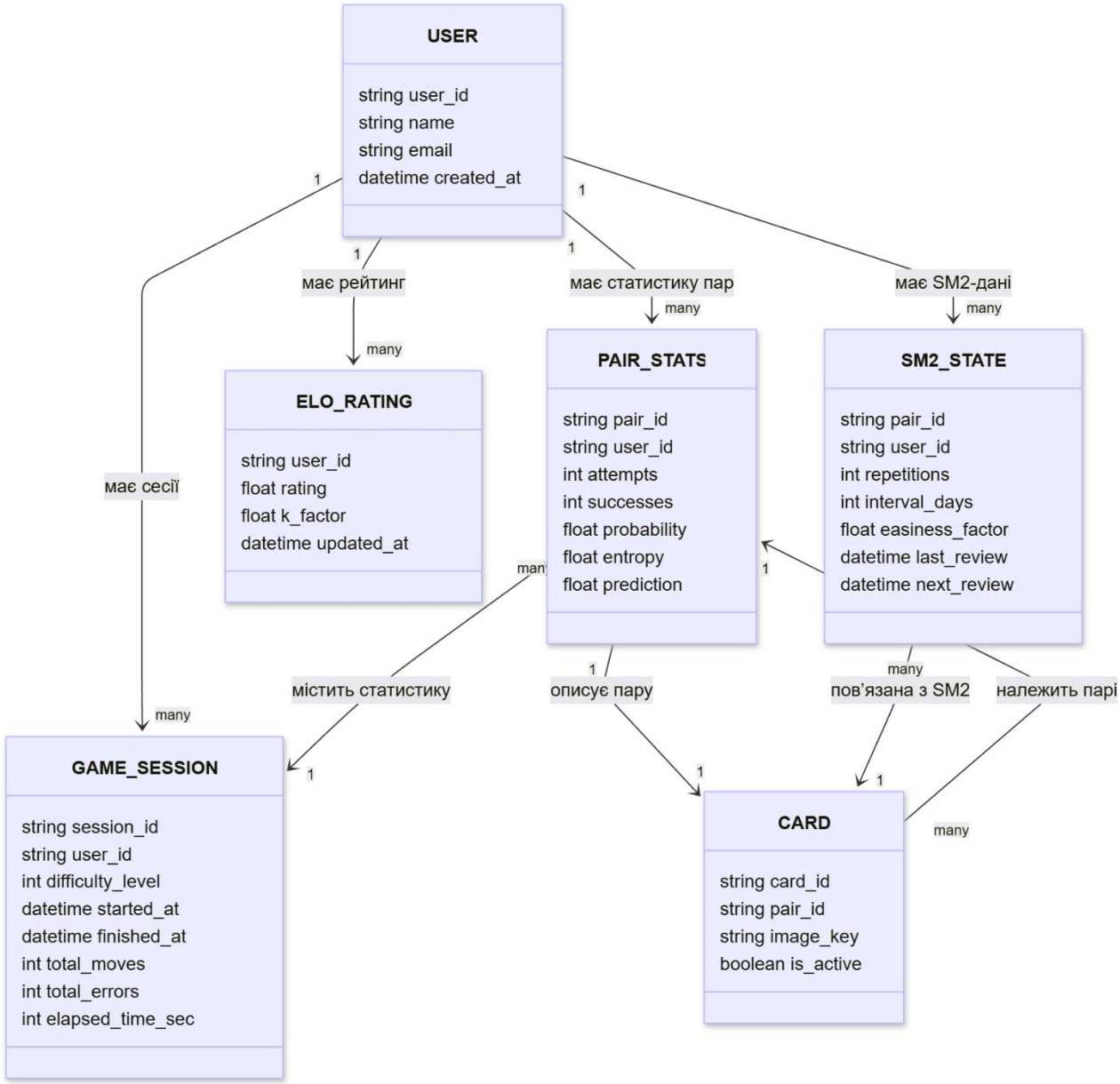


Рисунок 2.4 – Логічна ER-модель даних адаптивної системи Memory+

Таким чином, інформаційне забезпечення навчальної гри Memory+ включає комплекс структур, моделей та інформаційних процесів, що забезпечують функціонування системи та підтримку адаптивного навчання[26]. Правильна організація інформації є основою для подальшого математичного моделювання, розробки алгоритмів адаптації та створення програмної реалізації системи, яка буде розкрито у наступних розділах.

2.2. Інформаційна модель користувача та когнітивних параметрів

Інформаційна модель користувача є одним із центральних елементів інфологічного забезпечення системи Memory+, оскільки саме вона визначає структуру даних, необхідних для побудови персоналізованої траєкторії когнітивного розвитку. У контексті адаптивної навчальної гри інформаційна модель має не лише описувати основні атрибути користувача, але й відображати його поведінкові характеристики, динаміку навчальних показників і статистичні параметри, які формуються під час взаємодії з ігровим середовищем.

Розроблення інформаційної моделі користувача ґрунтується на тих вимогах, що накладає адаптивна система навчання. На відміну від традиційних інформаційних систем, де дані про користувача обмежуються ідентифікаційними полями, у Memory+ модель значно ширша, оскільки включає кілька груп взаємопов’язаних інформаційних об’єктів: дані профілю, історію ігрових сесій, статистику виконання завдань, показники для адаптивних алгоритмів, параметри довготривалого запам’ятовування та індивідуальні налаштування.

Першим елементом інформаційної моделі є ідентифікаційні та профільні дані. Вони включають унікальний ідентифікатор користувача, ім’я (або логін), дату реєстрації, електронну пошту та пов’язані параметри, необхідні для збереження прогресу у серверній частині системи. Дані профілю формують основу для зв’язування користувача з його сесіями, рейтингами, індивідуальними параметрами та статистичними показниками.

Окрім базових атрибутів, профіль містить службову інформацію, яка використовується для авторизації, контролю доступу та забезпечення персоналізації інтерфейсу. Додатково зберігаються налаштування користувача, такі як вибраний режим адаптивності, конфігурація підказок, швидкість анімацій та інші параметри взаємодії із системою. Саме ці дані дозволяють платформі враховувати індивідуальні переваги, а також підтримувати цілісність і безперервність навчального досвіду під час переходу між різними пристроями або ігровими режимами.

Структура профільних даних побудована таким чином, щоб забезпечити швидкий доступ до найбільш важливих характеристик користувача, а також

підтримувати інтеграцію з аналітичними модулями. Завдяки цьому профіль виконує роль центральної сутності інформаційної моделі системи Memory+, навколо якої формуються всі інші компоненти даних.

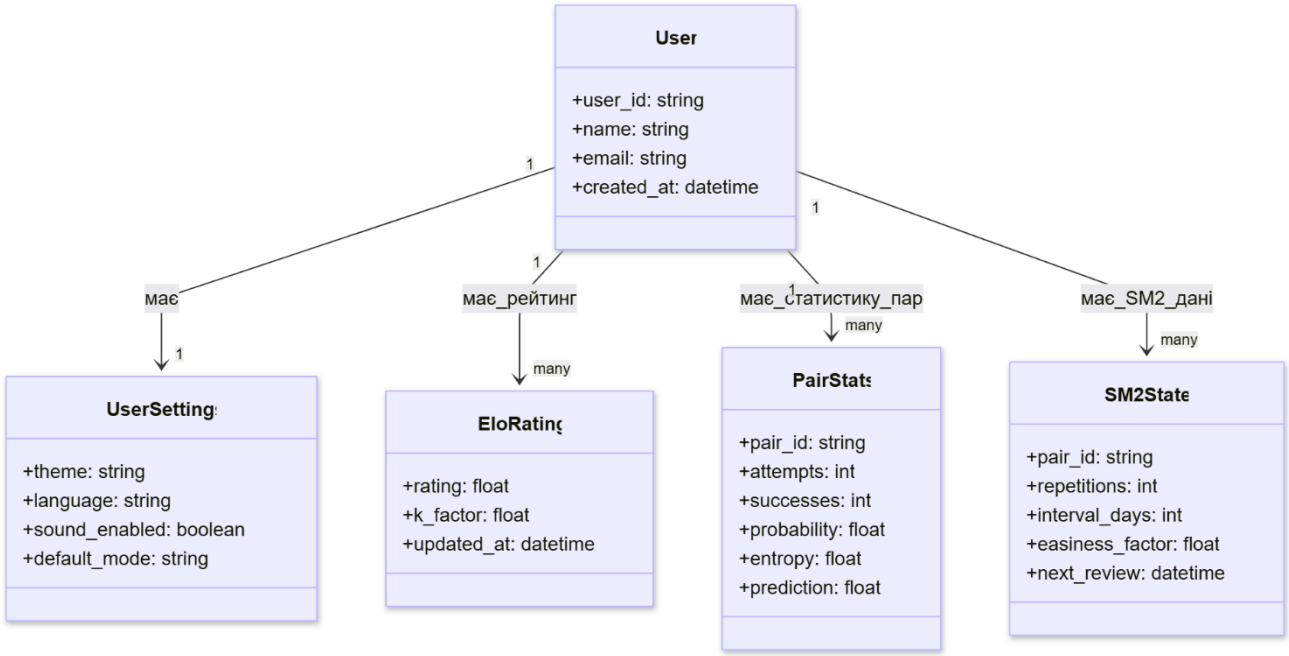


Рисунок 2.5 – Інформаційна модель користувача системи Memory+

Другим компонентом є структура ігрових сесій, що містить повну історію взаємодії користувача з грою. Кожна сесія має фіксований набір параметрів: дату та час початку, загальну тривалість, кількість ходів, кількість помилок, складність рівня, кількість відкритих пар, успішність виконання завдань та інші характеристики[25], які описують перебіг ігрового процесу. Дані сесії становлять основу для аналітичних алгоритмів та формують безперервний ланцюг навчального прогресу користувача. Крім того, структура сесій дозволяє відстежувати зміни у поведінці користувача та визначати динаміку розвитку його когнітивних навичок. На основі цих показників система приймає рішення про адаптацію рівня складності, забезпечуючи персоналізований підхід до навчання. Показники кожної сесії накопичуються та формують довгостроковий профіль користувача, що дає змогу алгоритмам точніше оцінювати його поточний рівень та прогнозувати майбутню успішність.

Також зібрані дані допомагають виявляти типові помилки, стратегії запам’ятовування та індивідуальний темп освоєння матеріалу, що суттєво підвищує ефективність аналітичних моделей. Завдяки цьому ігрові сесії виступають не лише

фіксацією процесу гри, а й ключовим джерелом інформації для подальшого удосконалення адаптивної системи.

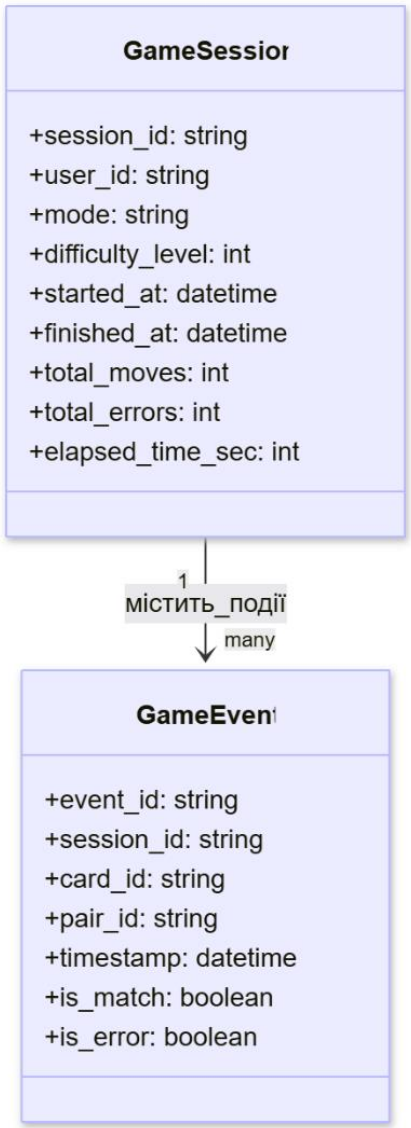


Рисунок 2.6 – Інформаційна структура ігрової сесії Memory+

Найважливішою частиною моделі користувача є статистичні та когнітивні параметри, що формуються на рівні окремих карткових пар. Для кожної пари система зберігає кількість спроб, кількість успішних відкриттів, оцінену ймовірність правильного вибору, ентропію та прогнозовані значення успішності, сформовані за допомогою експоненціального згладжування. Це дозволяє системі точно визначати, які саме елементи викликають труднощі, і з якою стабільністю користувач здатний їх запам’ятати. З огляду на те, що модуль microAnalysis.js обробляє події кожної

взаємодії, ці дані формуються в реальному часі та відображають актуальний когнітивний стан користувача.

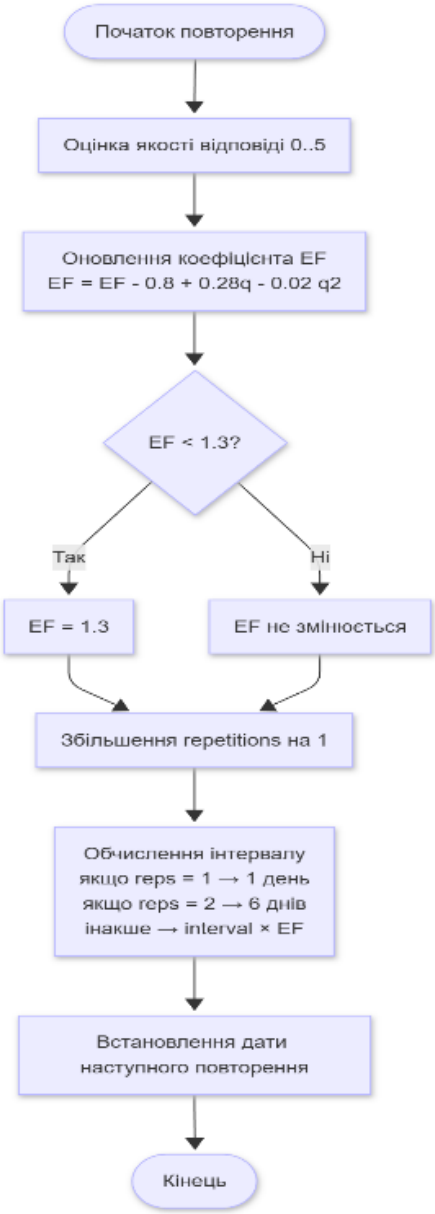


Рисунок 2.7 – Модель інтервального повторення SM-2 у системі Memory+

Четверта категорія даних - рейтингові параметри, сформовані за допомогою алгоритму Elo. Для кожного користувача система розраховує поточний рівень, що відображає загальну ефективність його дій у грі. Рейтинг оновлюється після кожної завершеної сесії, використовуючи співвідношення між очікуваною та фактичною результативністю. Такий підхід дозволяє забезпечити адаптивну зміну складності гри: чим вищий рейтинг користувача, тим більша кількість карток[11] і коротший час на

виконання завдання. Рейтингові параметри відіграють ключову роль у формуванні навчальної траєкторії.

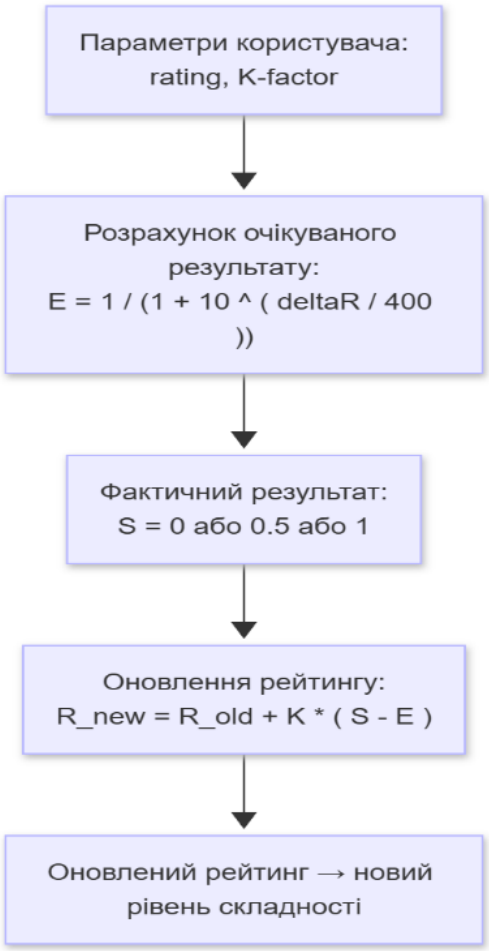


Рисунок 2.8 – Інформаційна структура рейтингів користувача (алгоритм Elo)

П’ятою групою є параметри довготривалого запам’ятовування, що формуються згідно з алгоритмом SuperMemo-2. Для кожної карткової пари фіксується кількість повторень, коефіцієнт легкості (EF), інтервал до наступного повторення та дата останнього показу. Ці параметри дозволяють системі не лише забезпечувати адаптацію в межах однієї сесії, але й будувати довготривалу стратегію навчання, спрямовану на перенесення інформації з короткочасної пам’яті у довготривалу.

Шоста частина інформаційної моделі - індивідуальні налаштування, що включають вибір мови, теми інтерфейсу, режиму гри, звукових ефектів та інших параметрів персоналізації. Хоча ці дані не впливають безпосередньо на алгоритми адаптації, вони формують комфорт користувача під час використання системи та забезпечують індивідуальний досвід взаємодії з ігровим середовищем.

Узагальнюючи, інформаційна модель користувача у Memory+ є складною багаторівневою системою, що включає як дані профілю, так і численні поведінкові показники, статистичні характеристики та когнітивні параметри, які формуються під час гри. Вона забезпечує інформативну основу для алгоритмів адаптації, дозволяючи формувати індивідуальну траєкторію навчання, оцінювати стан пам’яті користувача та оптимізувати процес повторення. Завдяки такій моделі система може виконувати функції когнітивного тренажера, інструмента розвитку пам’яті та адаптивної навчальної платформи.

2.3. Інформаційна структура ігрового середовища Memory+

Ігрове середовище системи Memory+ формується як сукупність інформаційних об’єктів, які взаємодіють між собою під час виконання навчального завдання. Структура цих об’єктів визначає логіку роботи гри, її адаптивні можливості, механіку генерування рівнів і правила обробки користувацьких дій. На відміну від класичних ігор типу Memory, де інформаційні елементи обмежуються набором карток і фіксацією співпадінь, у адаптивній системі ігрове середовище містить розширену інфологічну модель, яка забезпечує детальний збір даних і формування когнітивних показників.

Основою інформаційної структури є ігрове поле, яке формується на початку кожної сесії відповідно до рівня складності[4], визначеного алгоритмами адаптації. Ігрове поле складається з множини карток, кожна з яких має унікальний ідентифікатор, прив’язку до відповідної пари, графічне представлення та поточний стан (закрита, відкрита, деактивована після успішного співпадіння). Формування поля відбувається на основі параметрів користувача: його рейтингу, темпу навчання, стабільності виконання завдань і прогнозованої ефективності. Таким чином, набір карток і кількість пар не є фіксованими - вони змінюються залежно від когнітивного навантаження, яке система вважає оптимальним.

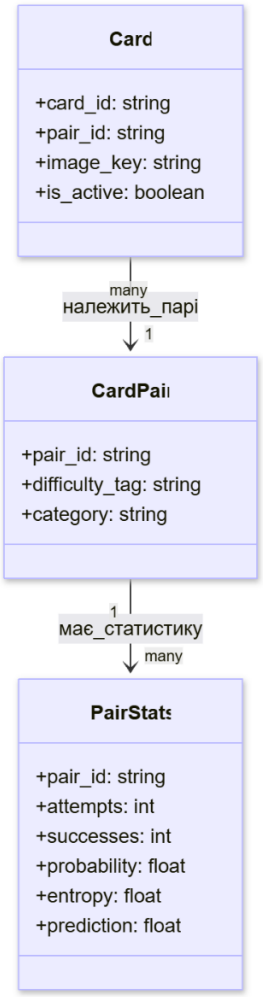


Рисунок 2.9 – Структура карток та пар у системі Memory+

Другим інформаційним компонентом є пари карток, які є ключовими одиницями навчального змісту. Для кожної пари формується пов’язана статистика, яка надалі використовується для аналізу запам’ятовування. Пара карток має власний ідентифікатор, групу графічних елементів та історію взаємодії з користувачем. Ця структура дозволяє системі відстежувати, наскільки добре користувач запам’ятовує конкретні об’єкти, та визначати, які елементи необхідно повторити або подати пізніше з меншим інтервалом.

Важливим інформаційним елементом є ігрові події (GameEvents) - вони описують кожну дію користувача: відкриття картки, співпадіння, помилку, затримку між діями, швидкість обробки інформації. Кожний GameEvent має часову мітку, параметри дії, ідентифікатори карток та зв’язок із відповідною сесією. Події формують основу для аналізу поведінкових патернів, виявлення труднощів та прогнозування динаміки результативності.

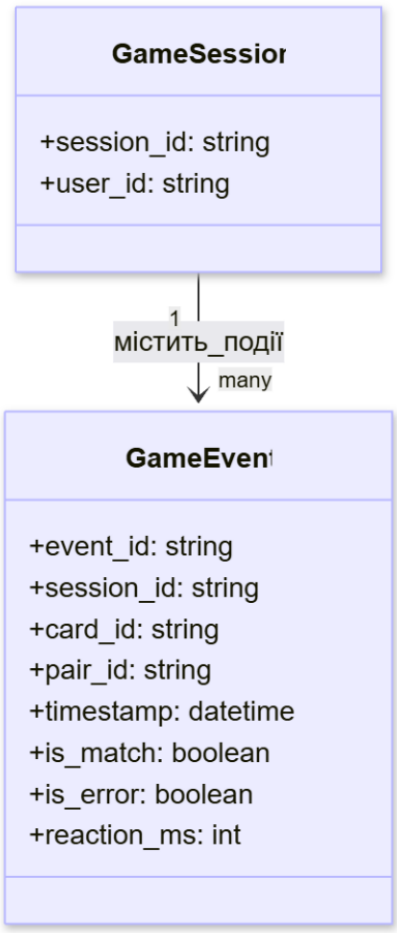


Рисунок 2.10 – Структура інформаційних подій у грі Memory+

Інформаційна структура також включає механізм побудови рівня складності, який визначає параметри ігрового поля: кількість карток, час на виконання, наявність підказок, швидкість оновлення станів та додаткові обмеження. Значення цих параметрів визначаються на основі даних із модулів microAnalysis, Elo та SM-2. Система об’єднує статистичні показники з історії попередніх рівнів і формує оптимальне когнітивне навантаження[8] відповідно до принципів адаптивного навчання.

Окремий інформаційний модуль займає структура сеансу (GameSession), яка пов’язує всі елементи ігрового середовища в єдину логічну систему. Сесія містить дані про користувача, параметри складності, історію подій, статистику успішності та підсумкові показники. У процесі гри GameSession безперервно оновлюється: змінюється кількість ходів, фіксуються правильні та помилкові дії, генерується граф поведінкових подій.

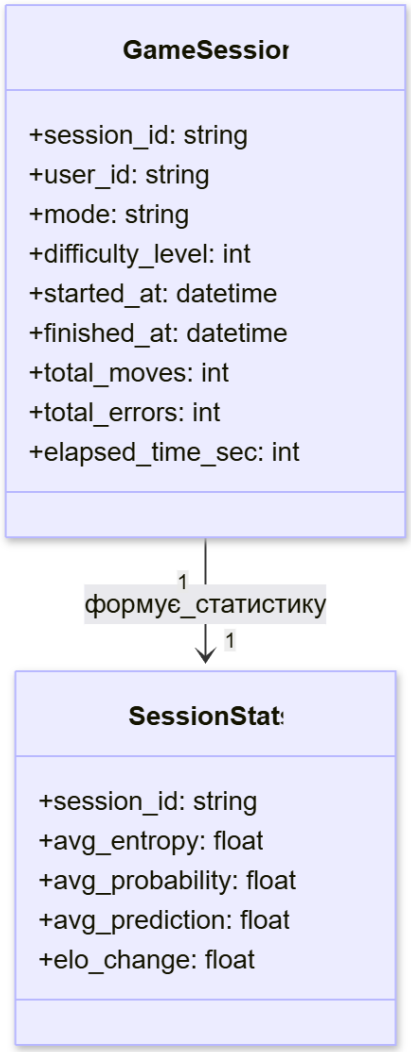


Рисунок 2.11 – Структура ігрової сесії Memory+

До інформаційної структури ігрового середовища належать також механізми формування навчальних сценаріїв, що визначають логіку подання завдань: порядок подання карток, частоту повторень, розташування елементів у полі та часові інтервали між діями. У традиційних реалізаціях Memory ці параметри є фіксованими; натомість у Memory+ вони визначаються динамічно, відповідно до когнітивних параметрів користувача. Наприклад, якщо ентропія для певних пар карток зростає, система може повторити ці пари частіше або зробити їх пріоритетними у наступній сесії.

Важливою складовою інформаційної структури є опис станів ігрових об’єктів, що включає поточні та транзитивні значення стану карток: закрита, відкрита, знайдена, заблокована, рекомендована до повторення. Ці параметри визначають поведінку інтерфейсу користувача та формують основу для побудови інтерактивності гри.

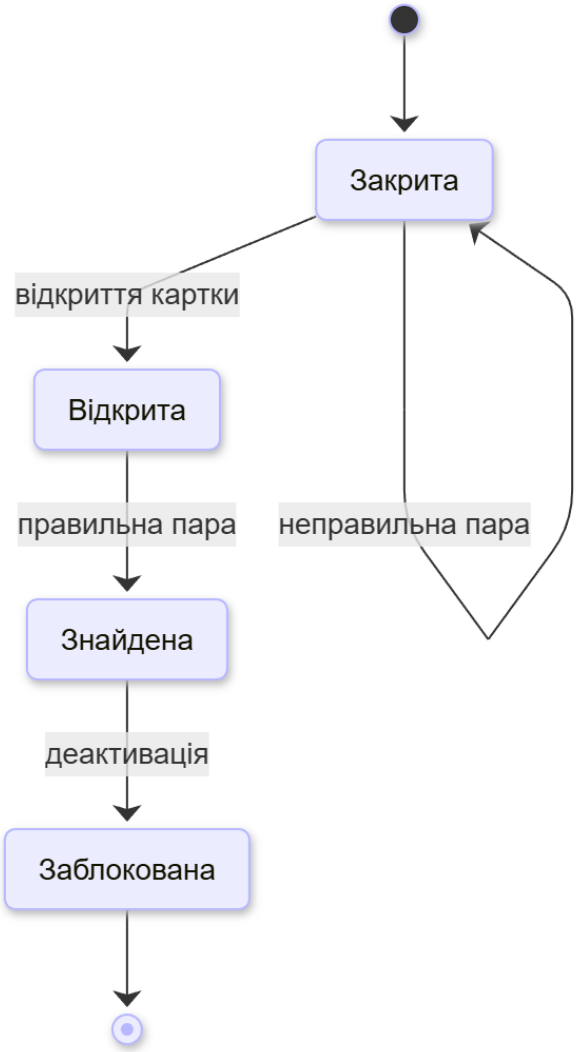


Рисунок 2.12 – Модель станів ігрових карток у Memory+

Загалом інформаційна структура ігрового середовища Memory+ є багаторівневою системою, яка об’єднує елементи навчального контенту, поведінкові дані та адаптивні параметри. Завдяки цьому ігрове середовище не лише забезпечує виконання навчального завдання, але й виступає платформою для збору даних, необхідних для аналізу когнітивних процесів та формування індивідуальної стратегії навчання. Складність структури виправдана тим, що кожен інформаційний компонент пов’язаний із алгоритмами адаптації, які будуть описані у наступному розділі.

2.4. Інформаційні процеси та потоки в адаптивній навчальній системі Memory+

Інформаційні процеси визначають динаміку обробки даних у системі Memory+ та є ключовим елементом інформаційного забезпечення адаптивної гри. На відміну від структурних моделей, які описують статику об’єктів, інформаційні процеси

відображають послідовність змін станів, обмін повідомленнями між модулями та логіку перетворення даних під час взаємодії користувача з ігровим середовищем. У цьому підрозділі розглядаються основні інформаційні потоки між компонентами Memory+, а також їх роль у формуванні когнітивних показників та адаптивної поведінки системи.

Першим етапом інформаційного процесу є ініціалізація ігрової сесії. Після авторизації або запуску гри система визначає початковий рівень складності, використовуючи поточні статистичні показники користувача, такі як рейтинг Elo, середня ентропія, прогнозована ефективність та стабільність виконання завдань. На основі цих даних формується ігрове поле: визначається кількість карток, їх розташування, набір пар та можливі обмеження часу. У цей момент відбувається активний обмін інформацією між модулями обчислення складності, статистики та генерації рівня.

Подальший інформаційний процес пов’язаний із виконанням ігрових дій, під час яких користувач взаємодіє з інтерфейсом гри. Кожна дія - відкриття картки, співпадіння або помилка - фіксується як окрема інформаційна подія з точним часовим маркером. Модуль обробки подій передає дані в підсистему мікроаналізу (microAnalysis), де відбувається обчислення ймовірності успішного відкриття пари, ентропії та прогнозу майбутніх результатів[22]. Ці процеси є безперервними і виконуються в реальному часі, забезпечуючи постійне оновлення інформаційного стану сесії.

Наступним інформаційним процесом є аналіз результатів і оновлення статистики. Модуль microAnalysis передає оновлені когнітивні показники до інших модулів: алгоритму Elo для оновлення рейтингу користувача та алгоритму SuperMemo-2 для оновлення параметрів довготривалого повторення. Цей обмін даними формує адаптивну поведінку системи: якщо користувач демонструє стабільно високі результати, система підвищує рівень складності; якщо виявляються труднощі з окремими картковими парами, алгоритм SM-2 планує повторні сесії з їхнім пріоритетним поданням.

Інформаційні процеси також включають формування рекомендацій та адаптивних рішень, які повертаються до ігрової логіки або інтерфейсу користувача. Наприклад, система може змінити час на виконання завдання, збільшити або зменшити кількість карток, адаптувати кольорові підказки або навіть визначити оптимальний момент для завершення сесії з урахуванням зниження когнітивної продуктивності. Усі ці рішення приймаються на основі інформаційних потоків між підсистемами статистики та адаптації.

Завершальним етапом інформаційного процесу є збереження інформації, яке може виконуватися як локально (у клієнтській частині), так і у серверній базі даних. Після завершення сесії система формує підсумкові показники: рейтинг, статистику, середні когнітивні значення та дані поведінкового аналізу. Ці параметри надходять до серверної частини, де зберігаються у структурованому вигляді для подальшого аналізу, використання у лідербордах або синхронізації між пристроями.

Особливістю системи Memory+ є те, що всі інформаційні процеси побудовані за принципом циклічної адаптації, коли результати поточної сесії визначають структуру наступної. Це робить інформаційні потоки не лінійними, як у більшості ігор, а рекурсивними, що дозволяє системі постійно вдосконалювати навчальний процес.

Узагальнюючи, інформаційні процеси в системі Memory+ становлять комплекс взаємодій між користувачем, підсистемою ігрової логіки, аналітичними модулями та серверною частиною. Вони забезпечують динамічне формування когнітивних показників, адаптивність і персоналізацію грального процесу, що є основою ефективного тренування пам’яті й уваги.

2.5. Інфологічна модель даних та організація зберігання інформації

Організація зберігання інформації у системі Memory+ є ключовим елементом інформаційного забезпечення, оскільки саме вона визначає, у якому вигляді дані про користувача, ігрові події, статистику та когнітивні показники зберігаються, передаються та обробляються. Складність системи, що базується на адаптивних алгоритмах, потребує ретельно побудованої інфологічної моделі, яка забезпечує цілісність, узгодженість і доступність інформаційних ресурсів. В основі Memory+

закладена клієнт-серверна архітектура, у якій зберігання даних може відбуватися як локально, так і у централізованому серверному сховищі.

Інфологічна модель системи описує сутності, які відіграють основну роль у когнітивному тренажері: користувачі, ігрові сесії, картки, пари карток, статистичні параметри, рейтингові дані, показники інтервального повторення та інформаційні події. Кожна сутність має власний набір атрибутів і зв'язків, які визначають її функціональне призначення у рамках системи. Такі структури забезпечують можливість відтворення повної історії навчальної взаємодії, а також використання накопичених даних для математичного аналізу та адаптації рівнів складності.

Базовим об'єктом інфологічної моделі[24] є користувач, який має ідентифікаційні дані, індивідуальні налаштування, рейтингові параметри та статистичні значення. Наступною сутністю є ігрова сесія, яка об'єднує всі дії користувача в межах однієї взаємодії з системою. Дані про сесію включають час початку та завершення, загальну тривалість, рівень складності, кількість ходів, кількість помилок та інші показники, що описують поведінку користувача під час виконання завдання.

Важливою сутністю є інформаційні події, що відображають кожну дію користувача. Такі події необхідні для аналізу реакції, продуктивності та стабільності виконання завдань. Події формують поведінкову модель, яка використовується модулями microAnalysis, Elo та SM-2.

Найскладнішою сутністю є статистичні об'єкти, що зберігають когнітивні параметри користувача: кількість спроб, кількість успішних відкриттів, ймовірність успіху, ентропію та прогнозовану ефективність. Такі дані формуються у процесі виконання гри і мають бути узгодженими з усіма іншим підсистемами. Аналогічно зберігаються і параметри алгоритму SuperMemo-2, які визначають інтервали повторення та коефіцієнти легкості.

Центральне місце займає інфологічна ER-модель, що відтворює структуру всіх сутностей та зв'язків між ними. Ця модель є основою для подальшої реалізації фізичної структури бази даних у серверній частині.

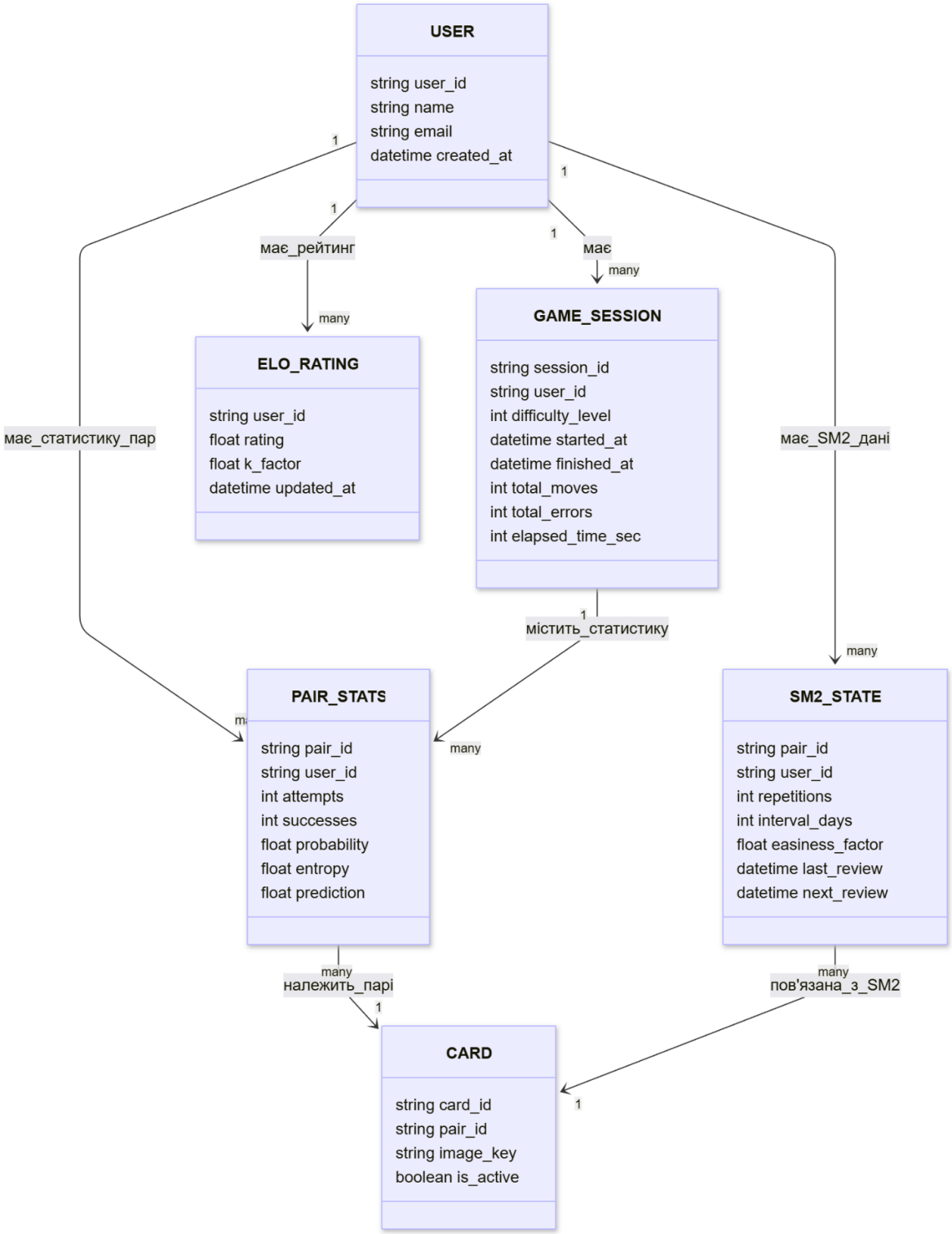


Рисунок 2.13 – Логічна ER-модель даних адаптивної системи Memory+

Організація зберігання інформації здійснюється на двох рівнях - клієнтському та серверному.

На клієнтському рівні (frontend) зберігаються тимчасові дані поточної сесії: стани карток, перелік подій, статистика активного рівня. Таке зберігання використовується для забезпечення швидкодії гри та мінімізації запитів до сервера.

Усі важливі дані (підсумкова статистика, рейтинги, параметри SM-2 та історія сесій) передаються у серверну частину і фіксуються в постійному сховищі.

На серверному рівні відбувається централізоване зберігання історичних даних, які є основою для аналізу динаміки користувача та побудови адаптивних алгоритмів. Серверна база даних містить таблиці користувачів, сесій, подій, статистики, рейтингів та інтервальних параметрів. Сервер також забезпечує узгодженість даних між різними клієнтськими пристроями та можливість входу у систему з будь-якого місця. Структура серверного API дозволяє системі передавати дані у форматі JSON, що забезпечує простоту інтеграції та гнучкість у побудові фронтенд-логіки.

Для забезпечення єдиної структури даних формується логічна JSON-модель, яка описує формат, у якому дані передаються між клієнтом і сервером. Така модель визначає структуру відповідей API та дозволяє стандартизувати об’єкти[17], що використовуються у комунікації між компонентами системи.

Рисунок 2.14 – Формат JSON-даних ігрової сесії у системі Memory+

JSON-модель може бути представленою як об’єкт із вкладеними структурами, що містять дані про сесію, події та статистику. Наприклад:

Такий підхід забезпечує прозорість даних, сумісність між компонентами системи та можливість подальшого розширення структури без порушення загальної архітектури.

Отже, інфологічна модель та структура зберігання даних у Memory+ формує основу функціонування адаптивної навчальної системи. Завдяки чітко побудованій моделі сутностей, добре структурованому серверному сховищу та стандартизованим форматам обміну даними, система може ефективно аналізувати поведінку користувача, зберігати історичні дані та формувати індивідуальні навчальні траєкторії. Цей підхід забезпечує масштабованість платформи та її подальший розвиток.

2.6. Організація зберігання даних та принципи побудови інформаційної архітектури системи

Організація зберігання даних у системі Memory+ є ключовим аспектом її функціонування, оскільки саме від цього залежить можливість відтворення історії навчальної взаємодії, забезпечення безперервності прогресу, реалізація адаптивного аналізу та формування персоналізованих навчальних траєкторій. На відміну від статичних ігор, де дані не мають тривалого життєвого циклу, в адаптивному навчальному середовищі інформація повинна зберігатися тривалий час, бути повною, узгодженою та доступною для математичної обробки.

Система Memory+ використовує комбінований підхід, який включає два основних рівні зберігання: клієнтський рівень (front-end storage) та серверний рівень (backend storage). Такий підхід дає змогу поєднувати високу швидкодію інтерфейсу з можливістю тривалого та централізованого збереження статистики, необхідної для аналізу когнітивного прогресу.

Першим рівнем є локальне зберігання даних, яке використовується під час виконання ігрової сесії. На цьому рівні система зберігає стан карток, набір подій, поточні показники результативності та проміжні статистичні значення. Таке зберігання є тимчасовим і забезпечує мінімальні затримки при взаємодії користувача з інтерфейсом. Локальні структури використовуються для оперативного аналізу, який не вимагає звернення до серверної частини.

Другим рівнем є серверне зберігання, яке відповідає за довготривале збереження даних. Воно містить інформацію про користувачів, історію сесій, статистику карткових пар, рейтингові значення, параметри інтервального повторення

та всі інші сутності, які є основою для адаптивної логіки. У серверній базі даних застосовується реляційний принцип організації інформації, що забезпечує цілісність даних, зв’язаність між сутностями та ефективний доступ до складних аналітичних вибірок[10].

Для забезпечення узгодженості між клієнтською та серверною частинами система використовує уніфікований формат обміну даними - JSON, який є стандартом для сучасних веб-застосунків. JSON-структури відповідають інфологічній моделі системи та включають вкладені об’єкти, які описують сесію, статистику, події та параметри адаптації. Використання такого формату дозволяє системі легко масштабуватися, додавати нові поля та змінювати структуру без порушення сумісності.

Комунікація між клієнтом і сервером здійснюється через REST API, який реалізує принципи CRUD-операцій (Create, Read, Update, Delete) для основних сутностей системи. Така архітектурна модель забезпечує простоту реалізації, передбачуваність процесів і можливість інтеграції з іншими сервісами, якщо система буде розширюватися.

На логічному рівні інформаційна архітектура Memory+ побудована у вигляді декількох взаємопов’язаних структур, які відповідають за окремі частини навчального процесу: структура користувача, структура ігрової сесії, структура подій, структура статистичних показників, структура даних адаптивних алгоритмів. Ця архітектура створює цілісну модель навчальної взаємодії, де кожен компонент має своє місце й призначення.

Для повноти опису інформаційної архітектури доцільно представити узагальнену схему зберігання та передачі даних у системі Memory+. Ця схема демонструє шлях інформації від моменту її формування (ігрова подія), її обробки аналітичними модулями[13] та передачі до серверної частини, де вона зберігається у вигляді структурованих сутностей.

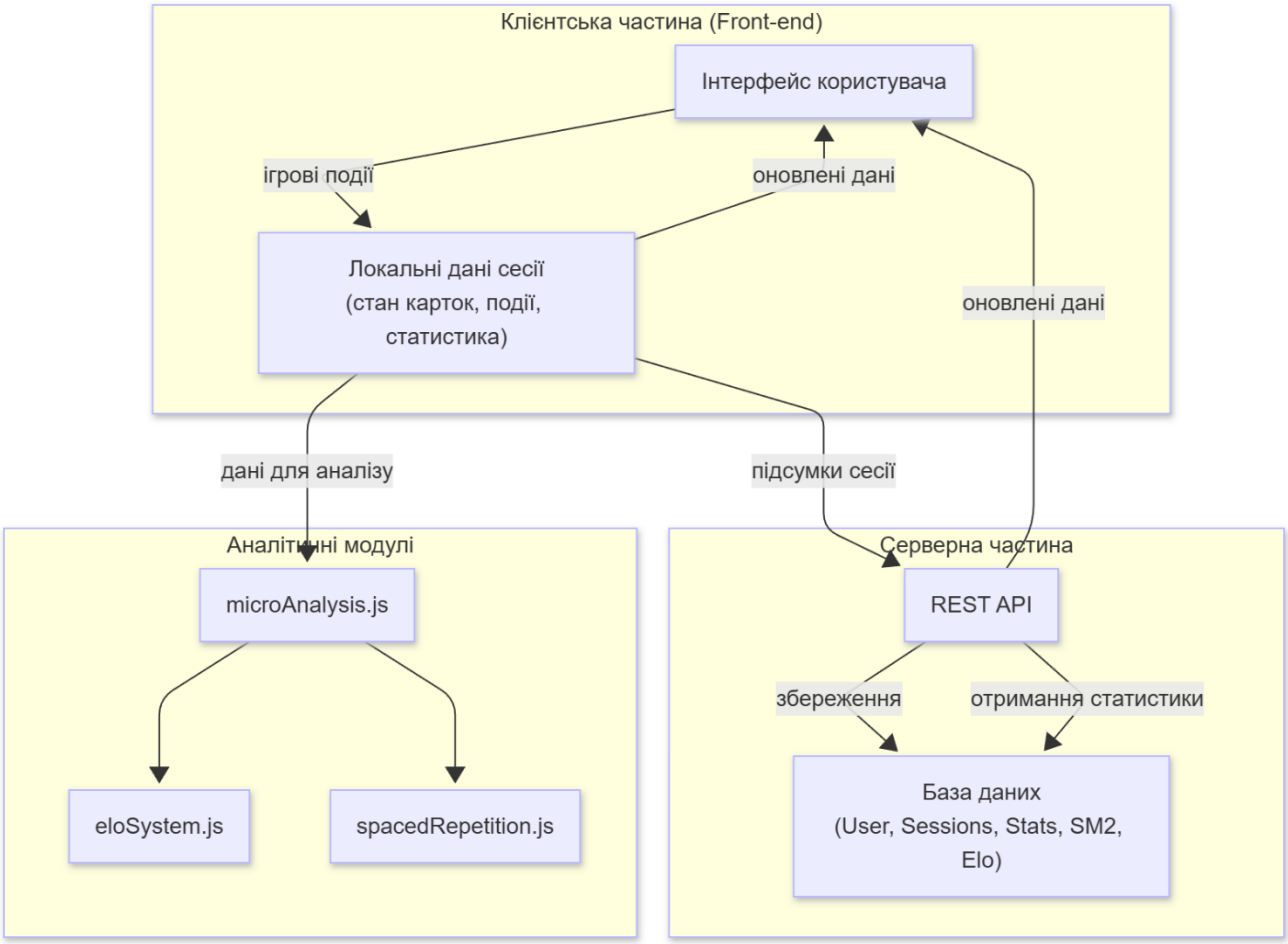


Рисунок 2.15 – Архітектура зберігання та обміну даними у системі Memory+

Завдяки такій організації даних система Memory+ має можливість формувати безперервний когнітивний профіль користувача, забезпечувати динамічний контроль складності, прогнозувати успішність виконання завдань і будувати довготривалу стратегію навчання. Добре структурована архітектура зберігання даних є критичним фактором, що забезпечує масштабованість[16] системи та підтримує реалізацію адаптивних алгоритмів, які описані в наступному розділі.

2.7 Висновки до розділу 2

У другому розділі було сформовано комплексне інформаційне забезпечення адаптивної навчальної системи Memory+, яке визначає зміст, структуру та організацію даних, необхідних для ефективної роботи когнітивного тренажера. Проведений аналіз дозволив сформувати інфологічну модель системи, що об’єднує дані про користувачів, ігрові сесії, статистичні показники, інтервали повторення та параметри адаптивних алгоритмів.

У межах підрозділу 2.1 визначено вимоги до інформаційного забезпечення та обґрунтовано роль інформаційних ресурсів у процесі формування індивідуальної траєкторії навчання. Підрозділ 2.2 містив опис інформаційної моделі користувача, яка включає ідентифікаційні, поведінкові, статистичні та когнітивні дані, необхідні для роботи алгоритмів адаптації. У підрозділі 2.3 проаналізовано структуру ігрового середовища, що охоплює картки, пари карток, події, ігрові сесії та моделі станів, які визначають логіку взаємодії між елементами системи.

У підрозділі 2.4 наведено характеристику інформаційних процесів, які забезпечують обмін даними між інтерфейсом користувача, аналітичними модулями та серверною частиною. Ці процеси формують основу механізму адаптації, що дозволяє системі динамічно регулювати складність завдань та підлаштовуватися під рівень користувача. У підрозділі 2.5 було розглянуто інфологічну модель даних[4] та принципи побудови їхнього зберігання, а також представлено єдину інформаційну архітектуру, що забезпечує цілісність, узгодженість та доступність даних у клієнтсько-серверному середовищі.

В цілому, інформаційне забезпечення Memory+ є структуровано організованою системою взаємопов’язаних об’єктів, яка забезпечує повноцінний збір, зберігання та опрацювання навчальних даних. Його розроблення створює основу для математичного моделювання адаптивних алгоритмів, які розглядатимуться у наступному розділі, а також гарантує можливість масштабування та подальшого розвитку платформи як сучасного інструмента тренування когнітивних навичок.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Загальні положення математичного забезпечення адаптивних навчальних систем

Математичне забезпечення є фундаментальною основою побудови будь-якої адаптивної навчальної системи, оскільки саме математичні моделі визначають механізми аналізу поведінки користувача, прогнозування результатів, динамічної зміни складності та оптимізації навчального процесу. У контексті сучасних цифрових навчальних платформ математичні методи виконують не лише аналітичну функцію, але й забезпечують формування індивідуальної траєкторії навчання, що відповідає особливостям когнітивного стану користувача, його темпу роботи та рівню навантаження.

Адаптивні навчальні системи, на відміну від статичних, потребують безперервного оновлення моделі користувача та аналізу нових даних у реальному часі. Це означає, що математичне забезпечення має бути не лише точним, але й здатним обробляти поступово сформовані дані під час кожної взаємодії з користувачем. Зокрема, такі системи вимагають використання методів статистичного аналізу[8], інформаційної теорії[7], моделей прогнозування[11], оптимізаційних алгоритмів та рейтингових систем. Саме їх поєднання формує основу математичного апарату адаптивних платформ, включно з системою Memory+, яка реалізує комплексний підхід до тренування когнітивних функцій.

Однією з ключових вимог до математичного забезпечення є здатність кількісно оцінювати рівень користувача. Для цього використовуються рейтингові моделі, такі як алгоритм Elo, що традиційно застосовується у змагальних середовищах, але має високу ефективність і в навчальних системах. Він дозволяє визначити поточний рівень підготовки користувача та забезпечує коректну адаптацію складності за допомогою простої, але статистично обґрунтованої формули. Рейтингові моделі забезпечують отримання узагальненого числового показника, який може бути використаний як базовий елемент системи адаптації.

Іншою важливою складовою є моделі інформаційної теорії, а саме ентропія, яка використовується для аналізу невизначеності у поведінці користувача. Ентропія

дозволяє оцінити, наскільки стабільно користувач виконує завдання певного типу, і визначити, які елементи викликають найбільші труднощі. На відміну від рейтингової моделі, що оцінює загальний рівень, ентропія дозволяє здійснити локальний аналіз - визначити когнітивний статус кожної окремої карткової пари. Це дає змогу будувати точніші рекомендації та формувати структурований профіль запам'ятовування.

У системах, де необхідно враховувати динаміку розвитку навички, важливу роль відіграє експоненціальне згладжування, яке забезпечує прогнозування результативності користувача. Таке згладжування дозволяє враховувати часове зниження ваги старих даних і підсилення значущості нових спостережень. У контексті Memory+ прогноз використовується для визначення тенденції навчання: покращується користувач, чи навпаки, демонструє зниження ефективності[16]. На основі цього система може регулювати навчальне навантаження та визначати оптимальний момент для переходу до складніших рівнів.

Ще одним компонентом математичного забезпечення є алгоритми інтервального повторення, зокрема модель SuperMemo-2. Цей алгоритм використовується для побудови довгострокових стратегій повторення інформації, що забезпечує перенесення знань з короткочасної пам'яті у довготривалу. SM-2 враховує кількість повторень, якість виконання завдання, інтервали між повтореннями та коефіцієнт легкості запам'ятовування. Такі параметри дозволяють оптимізувати структуру навчального процесу та забезпечити високу ефективність тренування пам'яті.

Комбінація перелічених моделей створює комплекс математичних методів, які доповнюють один одного і забезпечують адаптивність навчальної системи. Рейтингова модель визначає загальний рівень користувача, ентропійний аналіз - складність окремих інформаційних елементів, згладжування - прогноз успішності, а алгоритм SM-2 - оптимальну стратегію повторення. У сукупності вони формують інтегровану математичну систему, що дозволяє Memory+ забезпечувати індивідуальний навчальний вплив.

Центральною вимогою до математичного забезпечення є можливість актуалізації моделі в реальному часі. Оскільки кожна дія користувача може змінити

його когнітивний стан, система повинна бути здатною негайно перераховувати рейтинги, ентропію, прогнози та інтервали повторення. Це висуває вимоги до швидкодії математичних алгоритмів і точності обчислень, які мають бути оптимізованими для роботи у браузерному середовищі та мобільних пристроях.

Узагальнюючи, математичне забезпечення Memory+ складається з низки моделей, які дозволяють точно оцінювати дії користувача, визначати його навантаження, прогнозувати зміну результативності та формувати індивідуальну траєкторію навчання. Саме математичні методи є тим ядром, яке робить адаптивну гру здатною до розвитку та інтелектуальної взаємодії. У наступних підрозділах буде детально розглянуто кожний з цих методів, їх використання у когнітивному тренажері та особливості застосування в системі Memory+.

3.2. Рейтингові моделі в адаптивних навчальних системах. Модель Elo та її модифікація для Memory+

Рейтингові моделі дозволяють здійснювати кількісну оцінку рівня користувача та регулювати складність навчальних завдань відповідно до його актуальної продуктивності. У системах адаптивного навчання рейтинг виступає інтегральним показником, який формується на основі поведінки користувача, його стабільності, кількості помилок, швидкості виконання завдань та динаміки розвитку навички. У Memory+ рейтинг на основі моделі Elo є центральним інструментом для автоматичної зміни складності рівня.

Модель Elo походить з шахового середовища, однак її математичні властивості дозволяють ефективно застосовувати її у навчальних та когнітивних системах. У Memory+ роль суперника користувача відіграє рівень складності, який сприймається моделлю як умовний опонент із власним “рейтингом складності”. Таким чином, система може формувати адаптивне навантаження, порівнюючи рейтинг користувача та рейтинг завдання.

Ключовим компонентом моделі Elo є формула очікуваної результативності. У Memory+ вона визначає, наскільки ймовірно користувач успішно виконає завдання за умови поточного співвідношення рівнів. Очікувана результативність задається співвідношенням:

$$E = \frac{1}{1+10^{(\frac{R_d - R_u}{400})}}, \quad (3.1)$$

де

E - очікувана ймовірність успіху,

R_d - рейтинг завдання (opponent rating),

R_u- рейтинг користувача.

Ця функція наближається до 0, коли завдання значно складніше за поточні можливості користувача, і до 1 - коли завдання є значно легшим. Графічне зображення залежності результативності від різниці рейтингів доцільно представити у вигляді окремого рисунка, який ілюструє нелінійну природу функції.

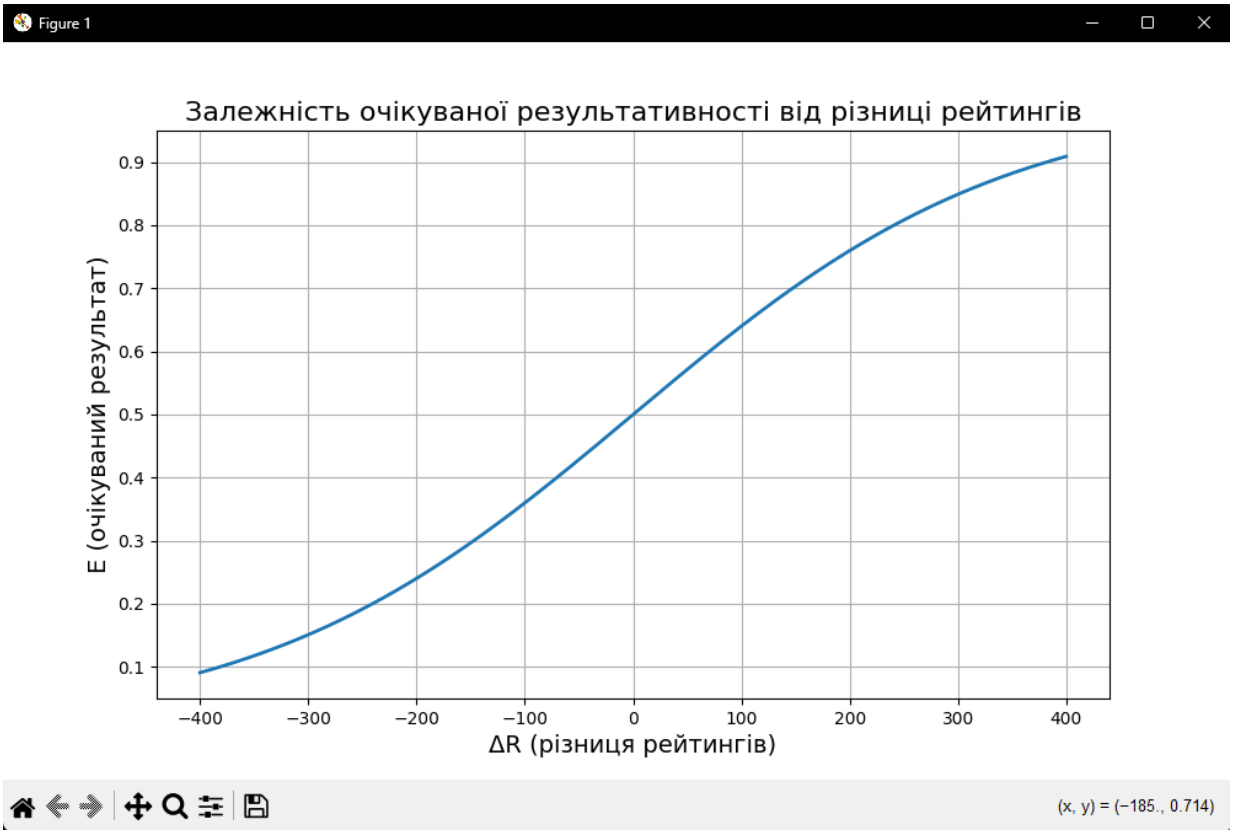


Рисунок 3.1 – Залежність очікуваної результативності від різниці рейтингів

Після завершення сесії система оновлює рейтинг користувача. Модель Elo пропонує простий та ефективний спосіб коригування рейтингу на основі очікуваного та фактичного результатів. У Memory+ оновлення рейтингу здійснюється за формулою:

$$R_u^{new} = R_u^{old} + K(S - E), \quad (3.2)$$

де

S - фактичний результат сесії,

E - очікуваний результат,

K - коефіцієнт адаптації.

У класичних іграх S визначається як 1 (виграш), 0.5 (нічия), 0 (програш).

У Memory+ результат є складнішим і розраховується на основі кількох параметрів:

$$S = w_1\left(1 - \frac{e}{m}\right) + w_2\frac{T_{opt}}{T_{real}} + +w_3 p_{success}, \quad (3.3)$$

де

S - інтегральний показник результативності,

w_1, w_2, w_3 - вагові коефіцієнти компонент оцінювання,

e - кількість допущених помилок,

m - загальна кількість ходів,

T_{opt}- оптимальний (еталонний) час проходження,

T_{real} - фактичний час проходження,

$p_{success}$ - ймовірність успіху користувача (модель прогнозу).

Це формує реалістичну оцінку ефективності користувача.

Для демонстрації процесу оновлення рейтингу наведено приклад, який може бути представлений у вигляді таблиці.

Таблиця 3.1 – Приклад оновлення рейтингу користувача

Параметр	Значення
Рейтинг користувача R_u	1050
Рейтинг рівня R_d	1100
Очікування E	0.43
Фактичний результат S	0.80
Коефіцієнт K	25
Новий рейтинг	1067.5

У Memory+ модель Elo не лише оцінює рівень користувача, а й виконує роль механізму регулювання складності. Після оновлення рейтингу система коригує параметри наступного рівня: кількість карток, обмеження часу, складність графічних образів, наявність підказок тощо.

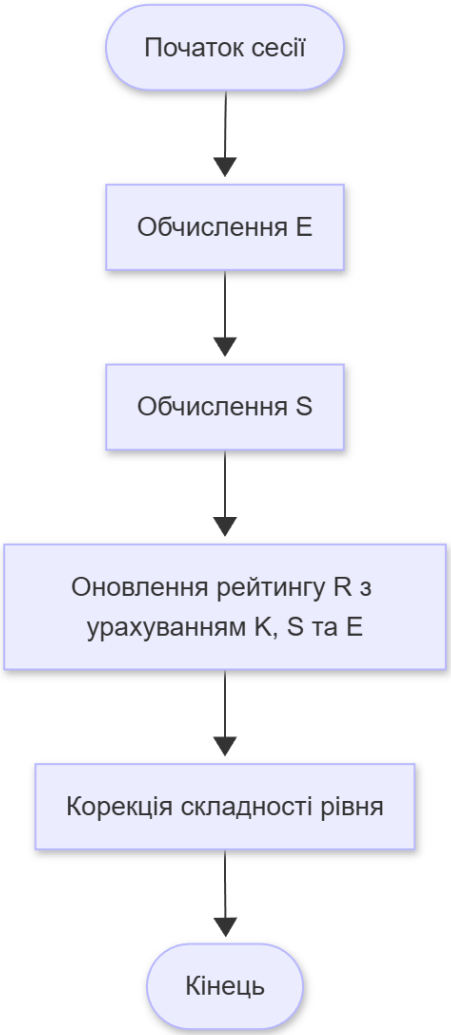


Рисунок 3.2 – Узагальнена схема роботи рейтингової моделі Elo у Memory+

3.3. Інформаційно-статистичні моделі оцінювання успішності користувача. Ентропія Шеннона та її застосування у Memory+

Ентропія є фундаментальним поняттям теорії інформації, яке характеризує міру невизначеності або непередбачуваності даних. У контексті адаптивних навчальних систем вона використовується для вимірювання складності завдання та стабільності результатів користувача. У грі Memory+, де кожна карткова пара представляє окремий когнітивний елемент, ентропія дозволяє визначити, які пари є легкими для запам’ятовування, а які - викликають стійкі труднощі.

У традиційних навчальних системах оцінювання результатів часто базується на кількості правильних відповідей або загальному часі виконання. Проте такі показники не завжди відображають реальний рівень невизначеності користувацьких дій. Ентропія дає змогу кількісно оцінити, наскільки передбачуваними є дії користувача під час відкриття певних карток. Якщо користувач часто помиляється, система фіксує високу ентропію, що означає нестійке запам'ятовування. Якщо ж користувач діє стабільно, ентропія буде низькою.

У Memory+ для кожної карткової пари розраховується ймовірність успішного відкриття p, яка визначається співвідношенням:

$$p = \frac{k_{success}}{k_{total}}, \tag{3.4}$$

де

$k_{success}$ - кількість успішних відкриттів,

k_{total} - загальна кількість спроб.

На основі цього значення обчислюється ентропія Шеннона, що визначається формулою:

$$H(p) = -p\log_2(p) - (1-p)\log_2(1-p), \tag{3.5}$$

де

$H(p)$ - ентропія інформації для результатів спроб,

k_{total} - ймовірність успішної дії користувача.

Ця формула описує кількість інформації, необхідної для передбачення наступної дії користувача. Ентропія досягає максимального значення, коли ймовірність успішної та невдалої дії однакова (p=0.5), і зменшується, коли поведінка стає більш передбачуваною (близько до 0 або 1). Висока ентропія свідчить про те, що користувач не має чіткої ментальної моделі для цієї карткової пари, тому її необхідно повторювати частіше.

Для наочності доцільно зобразити залежність ентропії від імовірності у вигляді графіка.

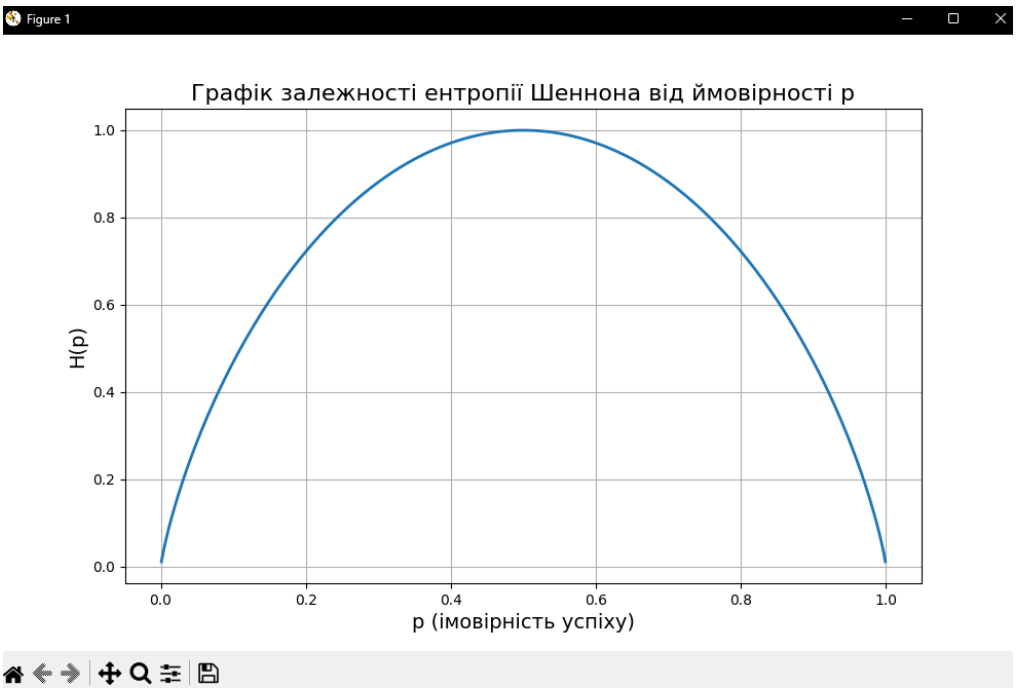


Рисунок 3.3 – Графік залежності ентропії Шеннона від ймовірності p

Значення ентропії у Memory+ є динамічними та залежать від поточної статистики користувача. Під час кожної взаємодії система оновлює кількість успішних і неуспішних відкриттів. Коли ентропія зростає, це свідчить про збільшення невизначеності, а отже, система може приймати рішення про необхідність повторення конкретних пар карток за алгоритмом SM-2 або зниження складності.

Для демонстрації обчислення ентропії наведемо приклад для різних значень ймовірності:

Таблиця 3.2 – Значення ентропії для різних імовірностей p

p	H(p)
0.1	0.47
0.3	0.88
0.5	1.00
0.7	0.88
0.9	0.47

У Memory+ ентропія не використовується окремо - вона є частиною комплексної моделі визначення складності. Наприклад:

- висока ентропія → пара подається частіше;
- середня ентропія → стандартне подання;
- низька ентропія → пара виноситься на пізніший етап повторення.

Таким чином, ентропія виступає локальним індикатором складності навчального матеріалу.

Окрім того, ентропія дозволяє виявити когнітивні патерни користувача. Якщо певні пари мають стабільно високу ентропію, це свідчить про:

- недостатнє засвоєння
- низьку асоціативність образів
- складність категорії
- проблеми короткочасної пам’яті

Для прикладу, у навчальній версії Memory+ (для шкіл) можна визначати, які типи образів (числа, літери, символи) даються дитині гірше.

У модулях мікроаналізу ентропія обчислюється автоматично після кожної взаємодії.

```
40  export function updatePairStats(pairStats, wasSuccessful) {
43
44    updated.attempts += 1;
45    if (wasSuccessful) {
46      updated.success += 1;
47    }
48
49
50    updated.probability = updated.attempts > 0
51      ? updated.success / updated.attempts
52      : 0.5;
53
54
55    const S_t = wasSuccessful ? 1 : 0;
56    updated.prediction = ALPHA * S_t + (1 - ALPHA) * updated.prediction;
57
58
59    if (updateSM2Stats) {
60      const updatedWithSM2 = updateSM2Stats(updated, wasSuccessful);
61      updated.sm2 = updatedWithSM2.sm2;
62    } else {
63
64      if (!updated.sm2) {
65        updated.sm2 = {
66          eFactor: 2.5,
67          repetition: 0,
68          interval: 1,
69          nextReviewDate: null,
70          lastReviewDate: new Date().toISOString(),
71          totalReviews: 0,
72        };
73      }
74    }
75
76    return updated;
77  }
```

Рисунок 3.4 – Фрагмент обчислення ентропії в модулі microAnalysis.js

3.4. Методи прогнозування результативності у адаптивних навчальних системах. Експоненціальне згладжування в Memory+

Одним з основних викликів адаптивних навчальних систем є необхідність не лише аналізувати історичні дії користувача, але й передбачати його подальшу успішність. Прогнозування дозволяє системі коригувати складність завдань заздалегідь, а не лише реагувати на минулі помилки. У когнітивних тренажерах, зокрема у Memory+, точне прогнозування має особливе значення, оскільки поведінка користувача може змінюватись у межах однієї сесії: від початкової плутанини – до стабільного виконання, або навпаки.

Одним із найефективніших методів прогнозування з низькою обчислювальною складністю є експоненціальне згладжування (Exponential Smoothing). Цей метод забезпечує динамічну оцінку очікуваної результативності користувача, приділяючи більше уваги новим спробам та поступово знижуючи вплив старих даних. Завдяки цьому модель реагує на будь-які короткочасні зміни у поведінці користувача та коректно оцінює тенденцію розвитку навички.

Експоненціальне згладжування: математична основа

Традиційне експоненціальне згладжування визначається формулою:

$$\widehat{p_t} = \alpha S_t + (1 - \alpha)\widehat{p_{t-1}}, \quad (3.6)$$

де

$\widehat{p_t}$- прогноз успішності в момент часу t

St - фактичний результат спроби (1 - успіх, 0 - помилка)

α - коефіцієнт згладжування (вага нових даних), $0 < \alpha < 1$

$\widehat{p_{t-1}}$ - попередній прогноз

У Memory+ використовується значення $\alpha = 0.25$ - це оптимальний баланс між реакцією на нові дані та стабільністю прогнозу.

Реалізація експоненціального згладжування у Memory+

```
export function getWeakPairs(pairsStats, threshold = 0.4) {
  const weakPairs = [];

  for (const [pairId, stats] of Object.entries(pairsStats)) {

    if (stats.attempts >= 2 &&
        (stats.probability < threshold || stats.prediction < threshold)) {
      weakPairs.push({
        pairId,
        stats,
        difficulty: (stats.probability + stats.prediction) / 2,
      });
    }
  }

  return weakPairs.sort((a, b) => a.difficulty - b.difficulty);
}

export function getStrongPairs(pairsStats, threshold = 0.7) {
  const strongPairs = [];

  for (const [pairId, stats] of Object.entries(pairsStats)) {
    if (stats.attempts >= 2 &&
        (stats.probability >= threshold || stats.prediction >= threshold)) {
      strongPairs.push({
        pairId,
        stats,
        strength: (stats.probability + stats.prediction) / 2,
      });
    }
  }

  return strongPairs.sort((a, b) => b.strength - a.strength);
}
```

Рисунок 3.5 – Обчислення прогнозу успішності методом експоненціального згладжування.

Поведінка алгоритму у контексті Memory+

На відміну від простих статистичних методів, експоненціальне згладжування дозволяє визначити:

- зниження уваги користувача у середині сесії,
- зростання впевненості у виконанні завдань,
- ефект втоми,
- нестабільні патерни поведінки,
- неочікувані провали після серії успіхів.

Якщо користувач помиляється вдруге поспіль після серії успішних дій, метод миттєво знижує прогнозоване значення. Якщо ж користувач діє стабільно, прогноз швидко сходиться до фактичної ймовірності.

Таким чином, згладжування дозволяє Memory+ визначати:

- чи слід знизити складність гри,
- чи потрібно запропонувати легший рівень карток,
- чи необхідно активувати SM-2 для подання слабших пар,
- чи підвищувати рівень складності шляхом збільшення кількості пар.

Графічна інтерпретація експоненціального згладжування

Для наочності корисно побудувати графік прогнозу на основі серії успіхів і помилок.

Цей графік показує, як швидко модель реагує на нові значення.

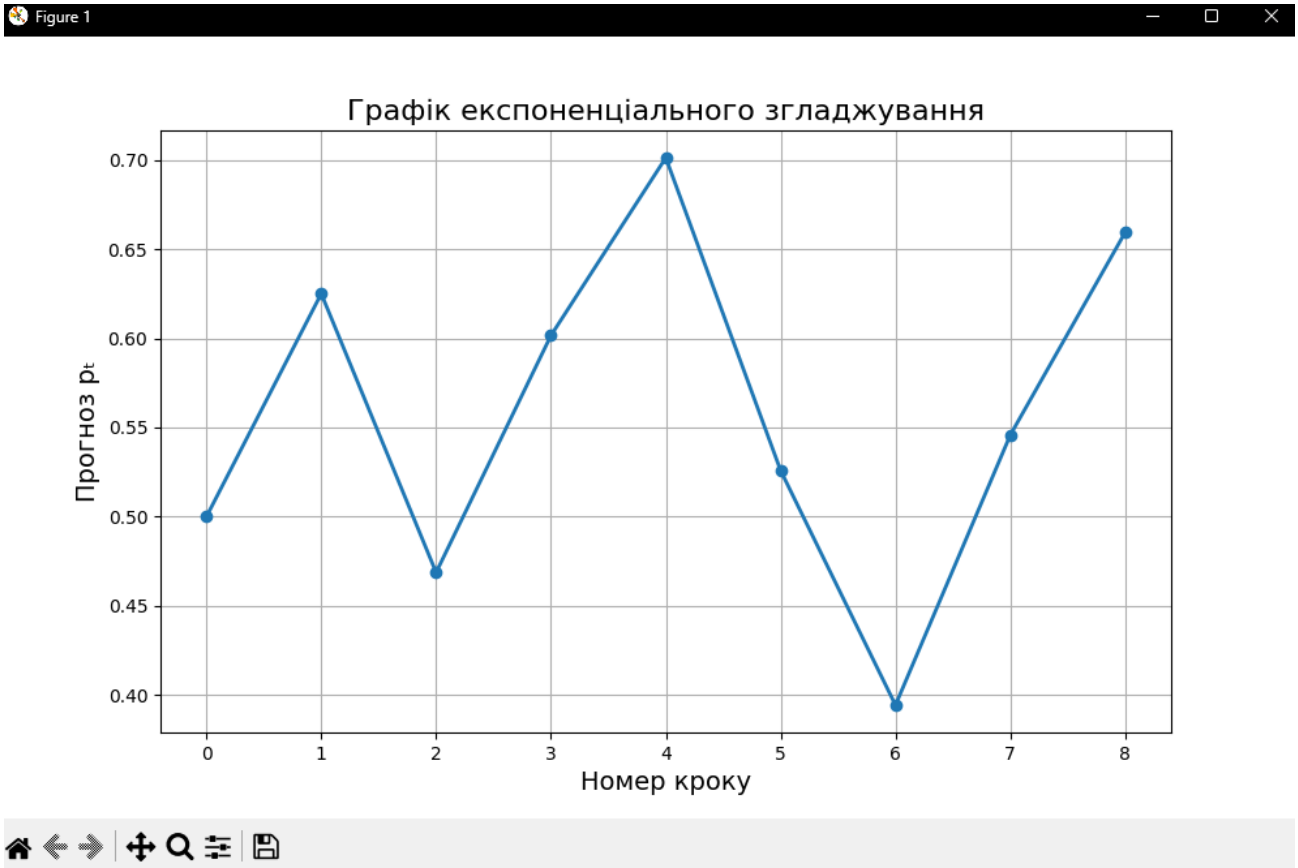


Рисунок 3.6 – Графік роботи експоненціального згладжування

Функціональна роль експоненціального згладжування в Memory+

У системі визначення рівня складності експоненціальне згладжування відіграє критичну роль.

Воно використовується для:

- прогнозу результативності
- визначення стабільності запам’ятовування
- порівняння користувацької поведінки з ентропією
- формування комплексного показника expectedScore

Функція: calculateExpectedScore() об’єднує

- ентропію
- нормалізовану ентропію
- середню ймовірність успіху
- прогнозовану результативність

і подає уніфікований показник очікуваної успішності, який використовується у виборі наступних рівнів та карткових пар.

Переваги експоненціального згладжування в адаптивних системах

Метод має низку переваг:

- висока чутливість до нових даних
- низька обчислювальна складність
- відсутність потреби зберігати великі масиви історії
- добре працює в реальному часі
- підходить для браузерних ігор

У Memory+ він є оптимальним способом прогнозування, оскільки система виконує десятки оновлень прогнозу в ході однієї ігрової сесії.

Таким чином, експоненціальне згладжування є ключовим елементом математичного забезпечення Memory+, оскільки забезпечує адаптивність системи, її здатність до прогнозування поведінки користувача та коректну реакцію на зміну рівня навичок у реальному часі. У наступному підрозділі буде розглянуто інтервальні алгоритми навчання, зокрема модель SuperMemo-2, яка формує довгострокову стратегію повторення інформації.

3.5. Моделі оптимального повторення інформації в адаптивних навчальних системах. Алгоритм SuperMemo-2 у Memory+

Ефективне навчання, що передбачає тривале запам’ятовування інформації, неможливе без коректного вибору моменту повторення навчального матеріалу. Базуючись на фундаментальних положеннях кривої забування Германна Еббінгауза, сучасні адаптивні системи використовують алгоритми інтервального повторення (spaced repetition), які мінімізують втрату інформації шляхом планування повторів у оптимальні моменти часу. Одним з найбільш досліджених та ефективних алгоритмів цього класу є SuperMemo-2 (SM-2), розроблений Пйотром Возняком.

Алгоритм SM-2 дозволяє навчальним системам формувати довгострокову модель запам’ятовування, адаптовану під конкретного користувача. У контексті Memory+ цей алгоритм стає не тільки механізмом для повторення, але і важливою частиною загальної системи адаптації, яка включає ентропійний аналіз, експоненціальне згладжування та рейтингову модель Elo.

Теоретичні засади інтервального повторення

Психологічні експерименти показують, що пам’ять не є рівномірною у часі: після первинного сприйняття інформації швидкість забування різко зростає, однак подальші повтори стабілізують слід у довготривалій пам’яті. Ця закономірність була математично описана у працях Еббінгауза, а пізніше розширена через адаптивні інтервальні моделі, які підлаштовуються під індивідуальні особливості користувача.

Алгоритм SM-2 формує інтервали повторення на основі:

- кількості повторів карткової пари;
- успішності кожного повторення;
- коефіцієнта легкості (EF), який характеризує, наскільки легко користувач запам’ятовує саме цю пару;
- адаптивного розширення інтервалів: від щоденних повторів на початкових етапах до повторів раз на декілька місяців.

У Memory+ кожна карткова пара має власну модель SM-2, що дозволяє системі адаптувати траєкторію навчання індивідуально для кожного користувача.

Математична модель роботи SuperMemo-2

Алгоритм складається з двох основних блоків: розрахунок інтервалів та оновлення коефіцієнта легкості.

1. Інтервали повторення

У класичному вигляді перші інтервали визначаються так:

$$I_1 = 1, I_2 = 6,$$

а всі подальші:

$$I_n = I_{n-1} \cdot EF, \tag{3.7}$$

де

I_1 - перший інтервал (1 день),

I_2 - другий інтервал (6 днів),

I_n - подальші інтервали множаться на EF.

Це означає, що пам’ять стабілізується експоненційно: чим краще користувач засвоїв матеріал, тим рідше система буде йому його повторювати.

У Memory+ це дозволяє:

- не навантажувати користувача легкими парами;
- зосередити увагу на складних елементах;
- зменшити втому та підвищити мотивацію.

Коефіцієнт легкості (EF)

EF визначається на основі оцінки успішності:

$$EF' = EF + \big(0.1 - (5 - q) * (0.08 + (5 - q) * 0.02)\big), \tag{3.8}$$

де

$q \in [0..5]$- якість згадування

Це забезпечує адаптацію інтервалів:

Таблиця 3.3 – Значення інтервалів для демонстрації

Кількість повторень	Оцінка q	EF	Інтервал I (дні)
1	5	2.5	1
2	5	2.5	6
3	4	2.36	14
4	3	2.02	28
5	5	2.02	56

Графік кривої повторень SM-2

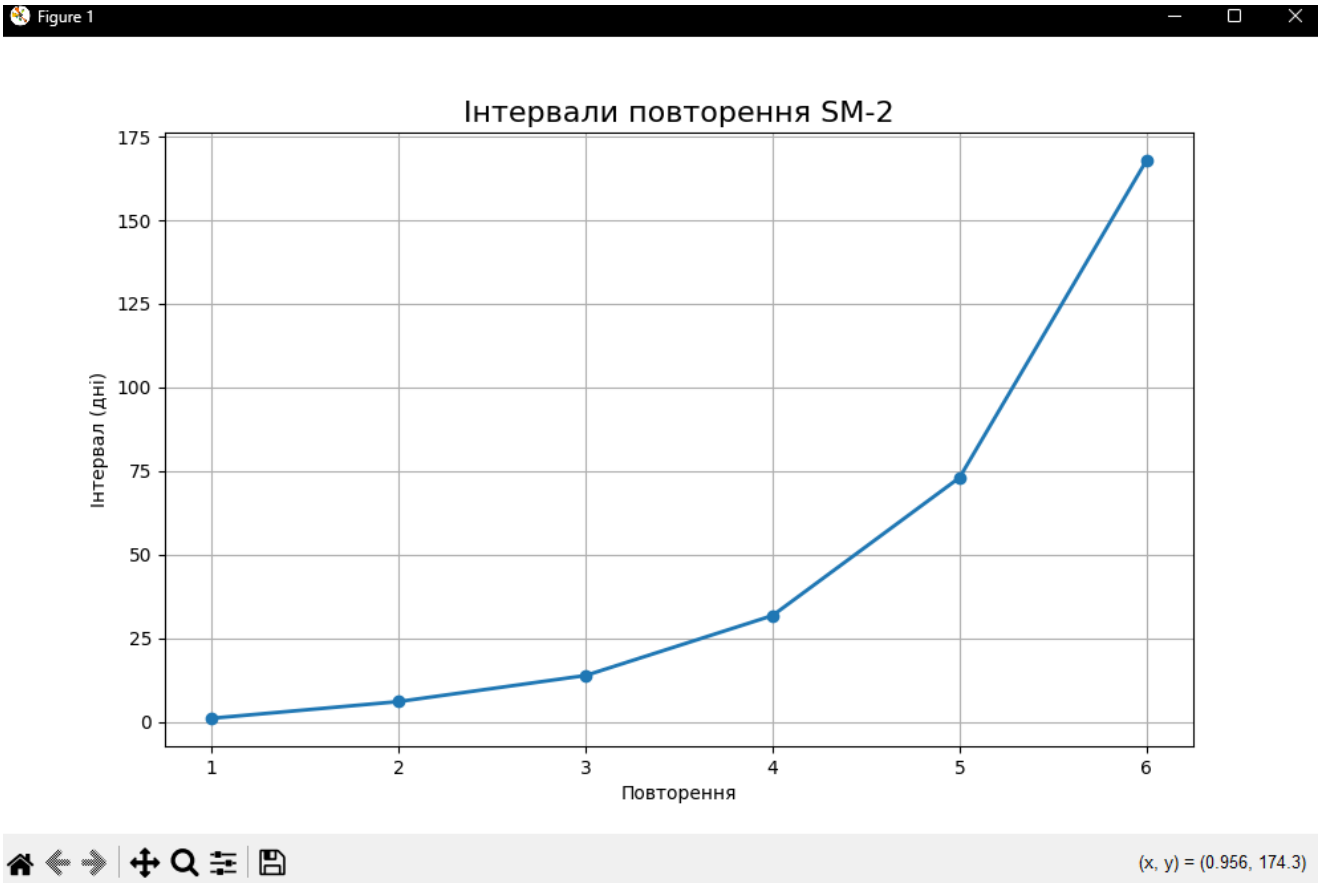


Рисунок 3.7 – Крива інтервалів повторення за алгоритмом SM-2

Практична інтеграція SM-2 у Memory+

На відміну від класичних програм для вивчення мов, Memory+ працює в режимі реального часу. Користувач виконує десятки мікродій під час однієї ігрової сесії, а система повинна швидко оновлювати статистику, не перевантажуючи браузер.

Ось ключовий фрагмент коду, який відповідає за інтеграцію SM-2 у мікроаналіз:

```
export function updateSM2Stats(pairStats, wasSuccessful, currentDate = new Date()) {
  const updated = { ...pairStats };


  if (!updated.sm2) {
    updated.sm2 = {
      eFactor: INITIAL_E_FACTOR,
      repetition: 0,
      interval: 1,
      nextReviewDate: null,
      lastReviewDate: currentDate.toISOString(),
      totalReviews: 0,
    };
  }

  const sm2 = updated.sm2;

  let quality;
  if (wasSuccessful) {

    if (updated.probability >= 0.9) quality = 5;
    else if (updated.probability >= 0.75) quality = 4;
    else quality = 3;
  } else {

    if (updated.probability >= 0.4) quality = 2;
    else if (updated.probability >= 0.2) quality = 1;
    else quality = 0;
  }

  sm2.eFactor = updateEFactor(sm2.eFactor, quality);


  if (wasSuccessful && quality >= 3) {
    sm2.repetition += 1;
    sm2.interval = calculateInterval(sm2.repetition, sm2.eFactor);
    sm2.nextReviewDate = calculateNextReviewDate(currentDate, sm2.interval).toISOString();
  } else {

    sm2.repetition = 0;
    sm2.interval = 1;
    sm2.nextReviewDate = calculateNextReviewDate(currentDate, 1).toISOString();
  }

  sm2.lastReviewDate = currentDate.toISOString();
  sm2.totalReviews += 1;

  return updated;
}
```

Рисунок 3.8 – Фрагмент SM-2 у коді

Його роль:

- формувати статистику sm2Stats для кожної карткової пари
- визначати інтервали наступного подання
- суміщати результати SM-2 з ентропією
- використовувати прогноз (prediction) для уточнення EF
- впливати на вибір слабких/сильних пар

Роль SM-2 у загальній адаптивній моделі Memory+

SuperMemo-2 не працює ізольовано - він є частиною комплексної математики, яка формується з таких компонентів:

1. Elo визначає поточний рівень складності, загальний "рейтинг" користувача.
2. Ентропія визначає локальну мінливість - які картки викликають складності.
3. Експоненціальне згладжування прогнозує короткочасні коливання успішності.
4. SM-2 визначає довготривалу траєкторію повторення.

Система Memory+ використовує всі ці показники для побудови збалансованого набору карткових пар у кожній сесії.

Тому SM-2 - це не просто «алгоритм повторення», а фундамент довготривалого планування навчання.

Алгоритм SuperMemo-2 є ключовим компонентом математичного забезпечення Memory+, оскільки формує оптимальні інтервали повторення для кожної карткової пари, враховуючи індивідуальні показники користувача. Інтегрований з ентропією, прогнозуванням та рейтинговою моделлю Elo, SM-2 забезпечує побудову унікальної траєкторії навчання, яка поєднує короткочасну адаптацію та довготривале закріплення знань.

Саме завдяки цьому Memory+ може виступати повноцінним тренажером для розвитку пам’яті й уваги - не на рівні одноразових вправ, а як систематична інтелектуальна платформа.

3.6. Інтегрована математична модель адаптивності в системі Memory+

У попередніх підрозділах були розглянуті окремі математичні компоненти, які реалізовані у системі Memory+: рейтинговий підхід на основі моделі Elo, інформаційно-статистичний аналіз (ентропія), експоненціальне згладжування результатів та інтервальний алгоритм повторення SuperMemo-2. Кожен із цих методів відповідає за певний аспект поведінки користувача, однак у реальній системі вони не існують окремо, а працюють спільно, утворюючи інтегровану модель адаптивності.

Метою цього підрозділу є формальне описання того, яким чином окремі математичні моделі поєднуються в єдину систему керування складністю, вибору навчального матеріалу та планування повторень у Memory+.

Загальні принципи інтеграції

Адаптивна система повинна одночасно:

- оцінювати поточний рівень користувача (глобальний рівень складності);
- аналізувати стабільність запам'ятовування окремих елементів (локальна складність);
- враховувати короткострокову динаміку (чи погіршуються/покращуються результати);
- планувати повторення в довгостроковій перспективі.

У Memory+ ці задачі розподілені наступним чином:

- модель Elo відповідає за оцінку загального рівня користувача;
- ентропія - за оцінку невизначеності та складності окремих пар;
- експоненціальне згладжування - за прогноз найближчих результатів;
- SuperMemo-2 - за довгострокову стратегію повторення карток.

На цій основі формується інтегрований показник складності, який використовується для прийняття рішень щодо:

кількості карток у рівні, часу на виконання, вибору слабких/сильних пар, повторення матеріалу.

Окремі складові інтегрованої моделі

1. Рейтинговий рівень користувача (модель Elo)

Оновлення рейтингу користувача після кожної сесії описується класичною формулою Elo.

$$R_{\text{new}} = R_{\text{old}} + K(S - E), \tag{3.9}$$

де

R_{old} – попередній рейтинг користувача,

R_{new} – новий рейтинг,

K – коефіцієнт чутливості,

S – фактичний результат,

E – очікуваний результат.

Очікуваний результат E обчислюється на основі різниці рейтингів користувача і поточного рівня складності.

2. Ентропія як міра невизначеності запам’ятовування

Для окремої карткової пари інформаційна ентропія визначається так.

$$H(p) = -p\log_2(p) - (1-p)\log_2(1-p), \tag{3.10}$$

де

p - ймовірність успішного відкриття.

Ця величина є максимальною при p=0.5 (повна невизначеність) і зменшується до нуля, коли користувач або майже завжди помиляється (p≈0), або майже не робить помилок (p≈1).

У Memory+ ентропія розраховується для всіх пар на основі накопиченої ймовірності успіху (поле probability в pairsStats). Функції calculateEntropy та normalizeEntropy у файлі microAnalysis.js реалізують цей підхід.

3. Короткостроковий прогноз результативності (експоненціальне згладжування)

У Memory+ використовується експоненціальне згладжування двійкового результату поточної спроби $S_t \in \{0,1\}$ для оцінки прогнозованої ймовірності успіху $\widehat{p_t}$.

$$\widehat{p_t} = \alpha S_t + (1-\alpha)\widehat{p_{t-1}}, \tag{3.11}$$

де

α - коефіцієнт згладжування (в грі використано значення α=0.25),

S_t - результат останньої спроби (1 - успіх, 0 - помилка),

$\widehat{p_{t-1}}$- попередній прогноз.

Ця формула прямо реалізована у файлі microAnalysis.js у фрагменті:

```
export function updatePairStats(pairStats, wasSuccessful) {
  const updated = { ...pairStats };

  updated.attempts += 1;
  if (wasSuccessful) {
    updated.success += 1;
  }


  updated.probability = updated.attempts > 0
    ? updated.success / updated.attempts
    : 0.5;


  const S_t = wasSuccessful ? 1 : 0;
  updated.prediction = ALPHA * S_t + (1 - ALPHA) * updated.prediction;


  if (updateSM2Stats) {
    const updatedWithSM2 = updateSM2Stats(updated, wasSuccessful);
    updated.sm2 = updatedWithSM2.sm2;
  } else {

    if (!updated.sm2) {
      updated.sm2 = {
        eFactor: 2.5,
        repetition: 0,
        interval: 1,
        nextReviewDate: null,
        lastReviewDate: new Date().toISOString(),
        totalReviews: 0,
      };
    }
  }

  return updated;
}
```

Рисунок 3.9 – Фрагмент експоненціального згладжування у коді

4. Довгострокова пам’ять та інтервали повторення (SuperMemo-2)

Алгоритм SM-2 визначає інтервали повторення на основі коефіцієнта легкості (EF) та кількості повторень. Перші два інтервали:

$I_1 = 1, \qquad I_2 = 6$

Подальші інтервали:

$$I_n = I_{n-1} \cdot EF, \tag{3.12}$$

де

I_n - інтервал перед n-тим повторенням,

EF - коефіцієнт легкості для конкретної пари.

Оновлення EF у класичному SM-2 відбувається на основі оцінки якості відповіді $q \in \{0, 1, 2, 3, 4, 5\}$. Стандартна формула:

$$EF' = EF + \big(0.1 - (5 - q)(0.08 + (5 - q) \cdot 0.02)\big), \qquad (3.13)$$

У Memory+ замість прямого введення оцінки q система може обчислювати її автоматично, використовуючи:

- фактичну ймовірність успіху (probability),
- прогнозовану ймовірність (prediction),
- кількість помилок та час реакції.

Реалізація SM-2 знаходиться у модулі spacedRepetition.js, а виклик його оновлення - у microAnalysis.js (рядок з updateSM2Stats).

```
55
56    if (updateSM2Stats) {
57      const updatedWithSM2 = updateSM2Stats(updated, wasSuccessful);
58      updated.sm2 = updatedWithSM2.sm2;
59    } else {
60
61      if (!updated.sm2) {
62        updated.sm2 = {
63          eFactor: 2.5,
64          repetition: 0,
65          interval: 1,
66          nextReviewDate: null,
67          lastReviewDate: new Date().toISOString(),
68          totalReviews: 0,
69        };
70      }
71    }
72
73    return updated;
74  }
```

Рисунок 3.10 – Виклик оновлення SM-2 у коді

```
178
179  export function selectPairsWithSM2(
180    allSymbols,
181    targetPairCount,
182    pairsStats,
183    eloRating,
184    currentDate = new Date()
185  ) {
186
187    const reviewCards = getCardsForReview(pairsStats, currentDate);
188    const strongCards = getStrongCards(pairsStats, currentDate);
189
190
191    const eloFactor = Math.min(1, (eloRating - 1000) / 1000);
192    const reviewRatio = 0.4 - eloFactor * 0.2;
193    const reviewCount = Math.max(1, Math.floor(targetPairCount * reviewRatio));
194
195    const selectedPairs = [];
196    const usedSymbols = new Set();
197
198
199    for (let i = 0; i < Math.min(reviewCount, reviewCards.length); i++) {
200      const card = reviewCards[i];
201      if (!usedSymbols.has(card.symbol)) {
202        selectedPairs.push(card.symbol);
203        usedSymbols.add(card.symbol);
204      }
205    }
206
207
208    const strongCount = Math.max(1, Math.floor(targetPairCount * 0.2));
209    for (let i = 0; i < Math.min(strongCount, strongCards.length); i++) {
210      const card = strongCards[i];
211      if (!usedSymbols.has(card.symbol)) {
212        selectedPairs.push(card.symbol);
213        usedSymbols.add(card.symbol);
214      }
215    }
```

Рисунок 3.11 – Реалізація SM-2 у коді

Інтегрований показник адаптації

Щоб приймати рішення про рівень складності, набір пар і параметри гри, Memory+ потребує узагальненого числового показника, який враховує всі вищенаведені чинники. У спрощеному вигляді такий показник складності можна записати як лінійну комбінацію нормованих параметрів:

$$C = \lambda_1(1 - E) + \lambda_2 H + \lambda_3(1 - \hat{p}) + \lambda_4 f_{\text{SM2}}, \qquad (3.14)$$

де

C - інтегрований показник складності для користувача на поточному етапі,

E - очікувана результативність за Elo (чим менше, тим важче завдання),

H - ентропія (чим більше, тим менш стабільна поведінка),

$\hat{p}$ - прогнозована ймовірність успіху (експоненціальне згладжування),

f_{SM2} - функція, що відображає «терміновість» повторення за SM-2 (наприклад, обернена величина до інтервалу або до різниці між поточною датою та nextReviewDate),

λ_1, λ_2, λ_3, λ_4 - вагові коефіцієнти, що визначають важливість кожного компонента.

Сенс формули:

– якщо очікувана результативність за Elo низька → $1-E$ зростає → система схиляється до зниження складності;

– якщо ентропія висока → H велика → система додає більше повторів або спрощує завдання;

– якщо прогнозована результативність низька → $1-\hat{p}$ велика → система зменшує навантаження;

– якщо за SM-2 настав час повторення (терміновість висока) → f_{SM2} зростає → система включає відповідну пару в поточну сесію.

Логіка прийняття рішень у Memory+

На основі показника C та окремих локальних метрик система приймає такі рішення:

- кількість пар у рівні:
- якщо C високе - кількість пар зменшується, якщо низьке - може бути збільшена;
- вибір слабких пар:
- ентропія та низька probability/prediction впливають на включення пари у список слабких;
- перемикання складності графічних образів:
- при високому рейтингу Elo та стабільній низькій ентропії можливе використання більш абстрактних або складних образів;
- включення пар за SM-2:

- пари, для яких за SM-2 настав час повторення, мають пріоритет у формуванні сесії.

У коді ці рішення реалізуються, зокрема, у функції selectPairsForGame, яка враховує:

- список усіх доступних символів (allSymbols),
- цільову кількість пар (targetPairCount),
- статистику (pairsStats),
- поточний рейтинг за Elo (eloRating),
- параметри SM-2 (useSM2, selectPairsWithSM2).

```
175  export function selectPairsForGame(
176    allSymbols,
177    targetPairCount,
178    pairsStats,
179    eloRating,
180    useSM2 = true
181  ) {
182
183    if (useSM2 && selectPairsWithSM2) {
184      return selectPairsWithSM2(allSymbols, targetPairCount, pairsStats, eloRating);
185    }
186
187    const eloFactor = Math.min(1, (eloRating - 1000) / 1000);
188    const weakPairsRatio = 0.2 + eloFactor * 0.2;
189    const weakPairsCount = Math.max(1, Math.floor(targetPairCount * weakPairsRatio));
190    const strongPairsCount = Math.max(1, Math.floor(targetPairCount * 0.4));
191
192    const weakPairs = getWeakPairs(pairsStats);
193    const strongPairs = getStrongPairs(pairsStats);
194
195    const selectedPairs = [];
196    const usedSymbols = new Set();
197
198    for (let i = 0; i < Math.min(weakPairsCount, weakPairs.length); i++) {
199      const symbol = weakPairs[i].pairId;
200      if (!usedSymbols.has(symbol)) {
201        selectedPairs.push(symbol);
202        usedSymbols.add(symbol);
203      }
204    }
205
206    for (let i = 0; i < Math.min(strongPairsCount, strongPairs.length); i++) {
207      const symbol = strongPairs[i].pairId;
208      if (!usedSymbols.has(symbol)) {
209        selectedPairs.push(symbol);
210        usedSymbols.add(symbol);
211      }
212    }
```

Рисунок 3.12 – Функція selectPairsForGame

Схема інтегрованої адаптації

Щоб наочно продемонструвати, як математичні моделі пов’язані між собою, доцільно вставити блок-схему.

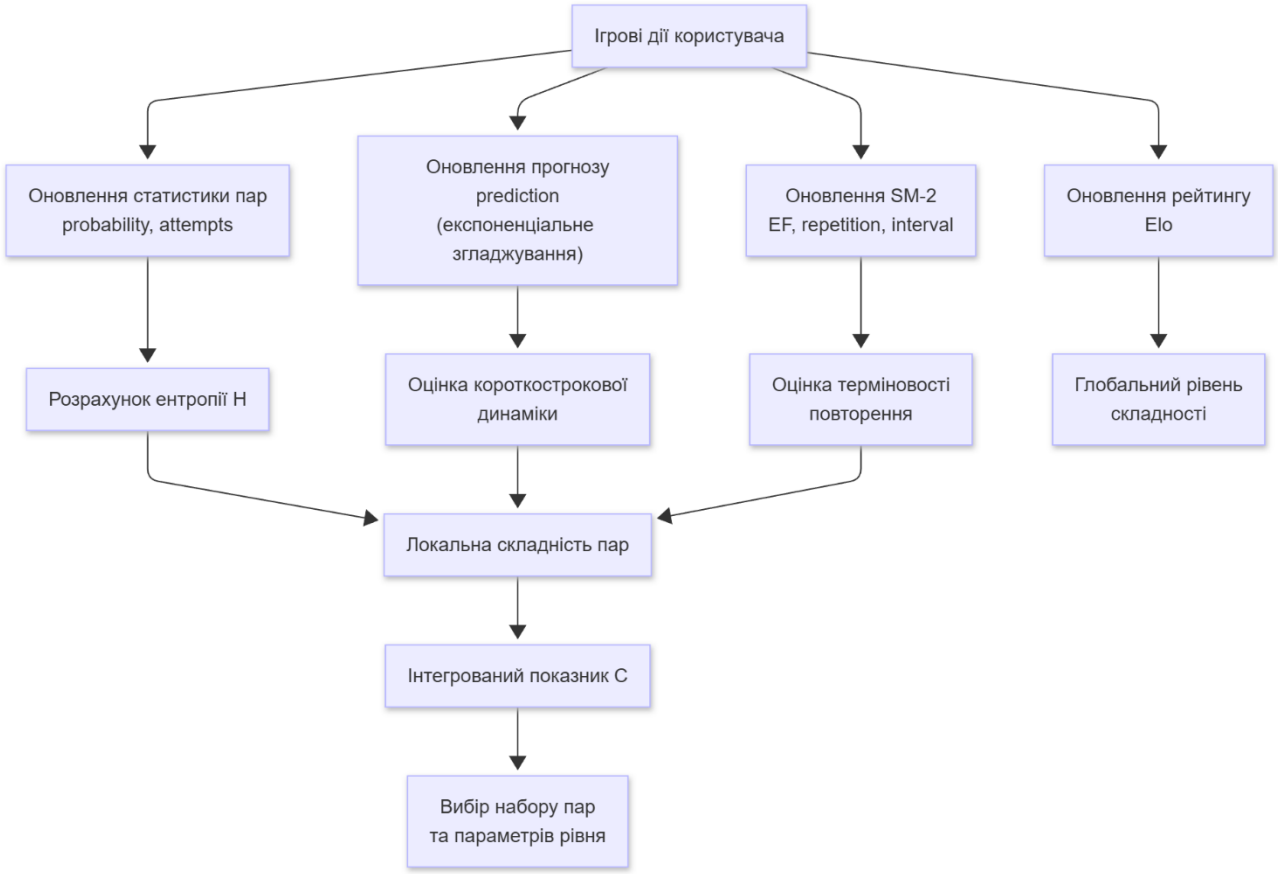


Рисунок 3.13 – Інтегрована модель адаптивності Memory+

Підсумкові зауваги

Інтегрована математична модель Memory+ забезпечує:

- кількісну оцінку загального рівня користувача (через Elo);
- виявлення проблемних елементів навчального матеріалу (через ентропію ймовірностей);
- прогнозування найближчих результатів (через експоненціальне згладжування);
- довгострокове планування повторення (через SM-2).

Саме завдяки поєднанню цих підходів система виходить за межі простої гри на запам’ятовування і перетворюється на адаптивний когнітивний тренажер, здатний

підлаштовуватися під індивідуальні особливості користувача, забезпечуючи більш ефективне й стійке формування навичок.

3.7. Висновки до розділу 3

У цьому розділі було розроблено та формально описано математичне забезпечення адаптивної системи Memory+. Було показано, що ефективна побудова індивідуальної траєкторії навчання потребує комплексного підходу, який поєднує моделі короткострокової і довгострокової пам’яті, статистичний аналіз поведінки користувача та самоадаптивну зміну рівня складності.

На основі рейтингової моделі Elo визначено механізм оцінювання загального рівня користувача, що дозволяє системі оперативно регулювати параметри складності у межах окремої ігрової сесії. Показано, що застосування ентропії Шеннона дає змогу формально оцінювати стабільність запам’ятовування окремих карткових пар і визначати ті елементи навчального матеріалу, які потребують додаткової уваги.

Окрему роль відіграє модель експоненціального згладжування, яка забезпечує короткостроковий прогноз результативності користувача. Цей підхід дозволяє виявляти тенденції покращення або погіршення пам’яті ще до того, як це позначиться на загальному рейтингу. Доповненням до цього є алгоритм SuperMemo-2, що забезпечує формування індивідуальних інтервалів повторення та сприяє перенесенню інформації у довготривалу пам’ять.

На завершення було сформовано інтегровану модель адаптивності, у якій усі математичні методи взаємодіють між собою та забезпечують узгоджене прийняття рішень. Така модель дозволяє Memory+ функціонувати не просто як гра з підбором пар, а як повноцінна адаптивна когнітивна система, здатна підлаштовуватися під особливості користувача, прогнозувати рівень успішності та підтримувати довготривале запам'ятовування.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1. Архітектура програмного забезпечення

Програмна система Memory+ побудована за принципами багаторівневої клієнт-серверної архітектури, що забезпечує чітке розділення між інтерфейсом користувача, прикладною логікою та шаром збереження даних. Такий підхід дозволяє досягти високої модульності, масштабованості та гнучкості у розширенні функціональності системи. Загальна структура Memory+ включає три основні рівні: фронтенд-застосунок, серверну частину (бекенд) та хмарну базу даних MongoDB Atlas.

Фронтендна частина реалізована як односторінковий застосунок на базі бібліотеки React. Вона відповідає за відображення ігрових інтерфейсів, обробку дій користувача, локальне збереження налаштувань та виконання запитів до серверного API. Логіка гри реалізована у вигляді окремих компонентів таких як MatchCardGame, LogicComboGame, SpeedRecallGame, GameBoard, Settings, ProfileStatistics та інші. Взаємодія між компонентами здійснюється через внутрішній стан React або контекст, що забезпечує узгоджений обмін даними в межах інтерфейсу.

Серверна частина побудована на платформі Node.js з використанням фреймворку Express. Основними її функціями є аутентифікація користувачів, обробка статистики гри, управління рейтингами, формування таблиць лідерів, обробка адаптивних даних та забезпечення взаємодії з базою даних. Бекенд реалізує REST API, яке забезпечує обмін даними у форматі JSON та підтримує повний цикл роботи користувача – від реєстрації та входу до оновлення особистої статистики й отримання рекомендацій.

Рівень збереження даних представлений хмарною базою MongoDB Atlas. Усі дані користувача, включаючи профіль, результати ігор, рейтинги, статистику адаптивних алгоритмів та параметри інтервального повторення (SM-2), зберігаються у колекції users. Для роботи з базою застосовується ODM-бібліотека Mongoose, яка забезпечує гнучке керування моделями даних та валідацію полів. Взаємодія з базою здійснюється через спеціальний модуль database.js, що реалізує підключення до кластера та обробку помилок.

Фронтенд і бекенд ізольовано один від одного та взаємодіють виключно через HTTP-запити до набору REST-ендпоїнтів. Це підвищує розширюваність системи та дозволяє у перспективі інтегрувати додаткові модулі, мобільні додатки або зовнішні сервіси. Авторизація реалізована за допомогою JSON Web Tokens (JWT), що дозволяє надійно перевіряти автентичність кожного запиту користувача.

Взаємодія компонентів системи може бути представлена у вигляді архітектурної схеми, на якій відображено роботу фронтенду, бекенду та бази даних, а також маршрути руху даних між ними.

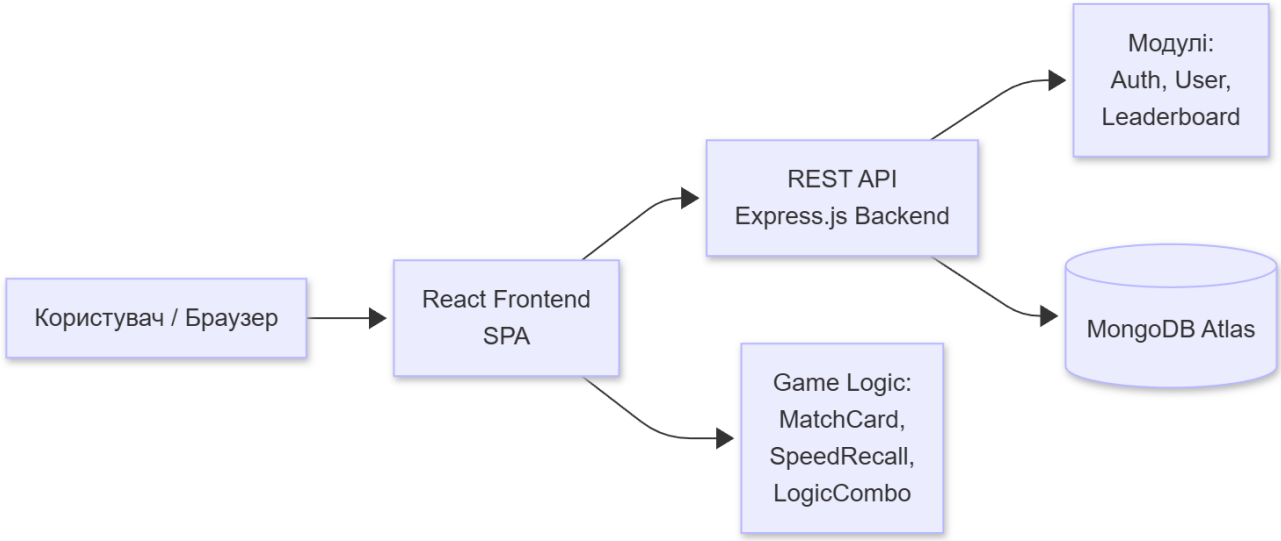


Рисунок 4.1 – Загальна архітектура програмної системи Memory+

Фронтенд з’єднується з бекендом через HTTP-запити до визначених маршрутів. Наприклад, Login.jsx надсилає POST-запит на /api/auth/login, після чого бекенд виконує перевірку через middleware auth.js, звертається до моделі User.js та повертає дані користувача разом із JWT-токеном. Бекенд виконує також запис статистики після кожної партії гри через маршрут /api/user/stats, який оновлює відповідні поля в User.js, включаючи рейтинги за Elo, статистику спроб, параметри SM-2 та інші адаптивні дані.

Окрім базових операцій, система підтримує низку сервісних ендпойнтів, що забезпечують доступ до історії сесій, розрахованих показників адаптації та зведених аналітичних метрик. Таким чином, фронтенд отримує актуальні дані в режимі реального часу і може оперативно оновлювати інтерфейс відповідно до поточного стану користувача. Комунікація між клієнтом і сервером побудована таким чином,

щоб мінімізувати затримки та запити до бази даних, використовуючи оптимізовані об’єкти відповіді та кешування ключових параметрів. Такий підхід забезпечує плавну роботу застосунку та створює передумови для масштабування, коли кількість активних користувачів зростає.

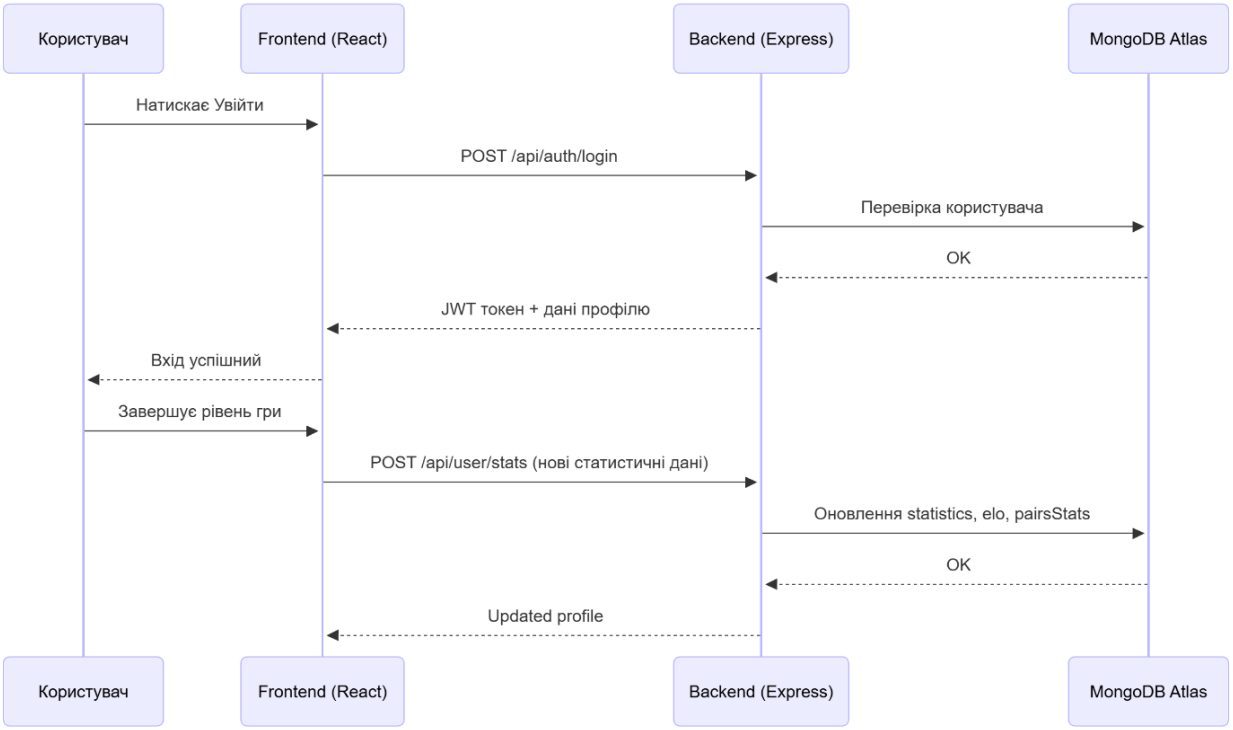


Рисунок 4.2 – Взаємодія фронтенду та бекенду через REST API

Окреме значення у системі має архітектура адаптивного модуля. Під час гри фронтенд збирає дані про дії користувача (успішність, час реакції, кількість спроб), після чого модулі microAnalysis.js, eloSystem.js та spacedRepetition.js формують набір статистичних параметрів: ймовірності успіху, прогнозовані значення, ентропію, оновлений рейтинг та SM-2 інтервали. Ці дані відправляються на бекенд, де зберігаються у полі pairsStats моделі User. У підсумку адаптивна логіка є розподіленою: обчислення виконуються на фронтенді, а довгострокове збереження забезпечує сервер.

Архітектура системи Memory+ є масштабованою і дозволяє легко додати нові ігрові режими, типи статистики або аналітичні алгоритми.. Docker-конфігурація реалізована у файлах Dockerfile, Dockerfile.frontend, Dockerfile.frontend-build та docker-compose.yml, що дозволяє запускати програмну систему як єдиний комплекс.

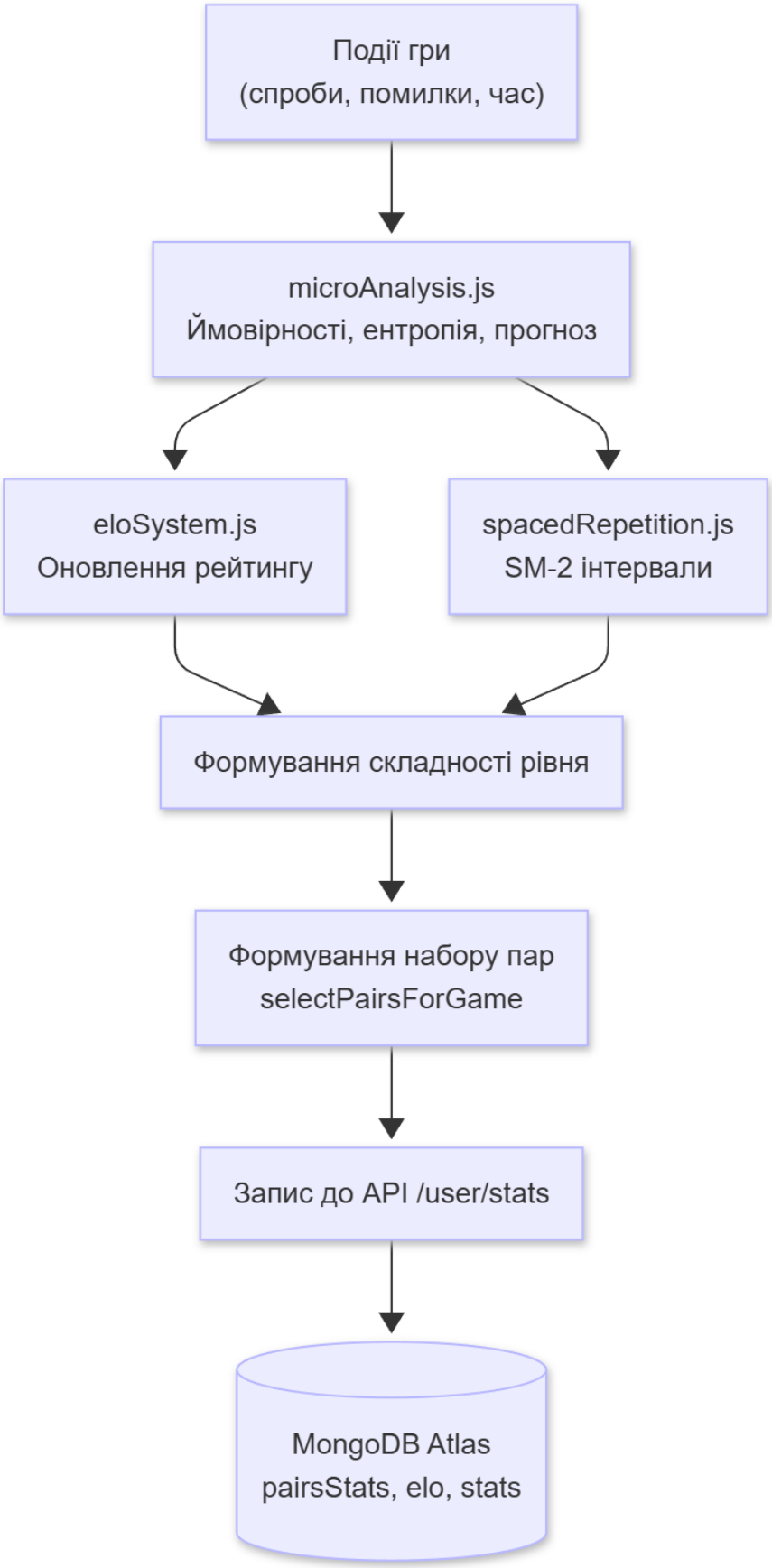


Рисунок 4.3 – Архітектура адаптивних алгоритмів у Memory+

Глибоке розділення на окремі архітектурні шари, модульність компонентів, використання сучасних веб-технологій та адаптивних алгоритмів робить Memory+ повноцінною інтерактивною системою тренування когнітивних здібностей, придатною до подальшого масштабування та дослідження в академічному та практичному середовищі.

4.2. Фронтендна частина системи

Фронтендна частина системи Memory+ реалізована у вигляді односторінкового веб-застосунку, побудованого на бібліотеці React. Такий підхід забезпечує високу продуктивність інтерфейсу, можливість динамічного оновлення даних без перезавантаження сторінки та чітку модульну структуру, що полегшує масштабування та подальше розширення функціональності системи. Компонентно орієнтована архітектура дозволила розмежувати логіку гри, інтерфейсні елементи, роботу з користувачем і модулі аналітики, зберігаючи при цьому єдину модель стану застосунку.

Загальна структура застосунку включає кілька функціональних груп компонентів: інтерфейс навігації, ігрові екрани, сервісні компоненти, профіль користувача та модулі налаштувань. Старт роботи системи відбувається через компонент App.jsx, який відповідає за ініціалізацію користувацьких даних, вибір активного екрану та передачу параметрів між окремими частинами застосунку. Відповідно до вибору користувача відображаються головне меню, меню міні-ігор, обраний ігровий режим, статистика або розділ налаштувань.

Архітектура компонентів побудована таким чином, щоб забезпечити максимальну модульність і повторне використання елементів інтерфейсу. Кожен екран складається з дрібніших компонентів — панелей, карток, кнопок керування та інформаційних блоків, — які динамічно оновлюються залежно від стану гри та дій користувача. Керування станом застосунку реалізоване через локальний State та контекстні механізми React, що забезпечує узгодженість даних між ігровою логікою та відображенням на екрані. Такий підхід підвищує масштабованість проєкту та спрощує інтеграцію нових режимів або функцій без порушення загальної структури системи.

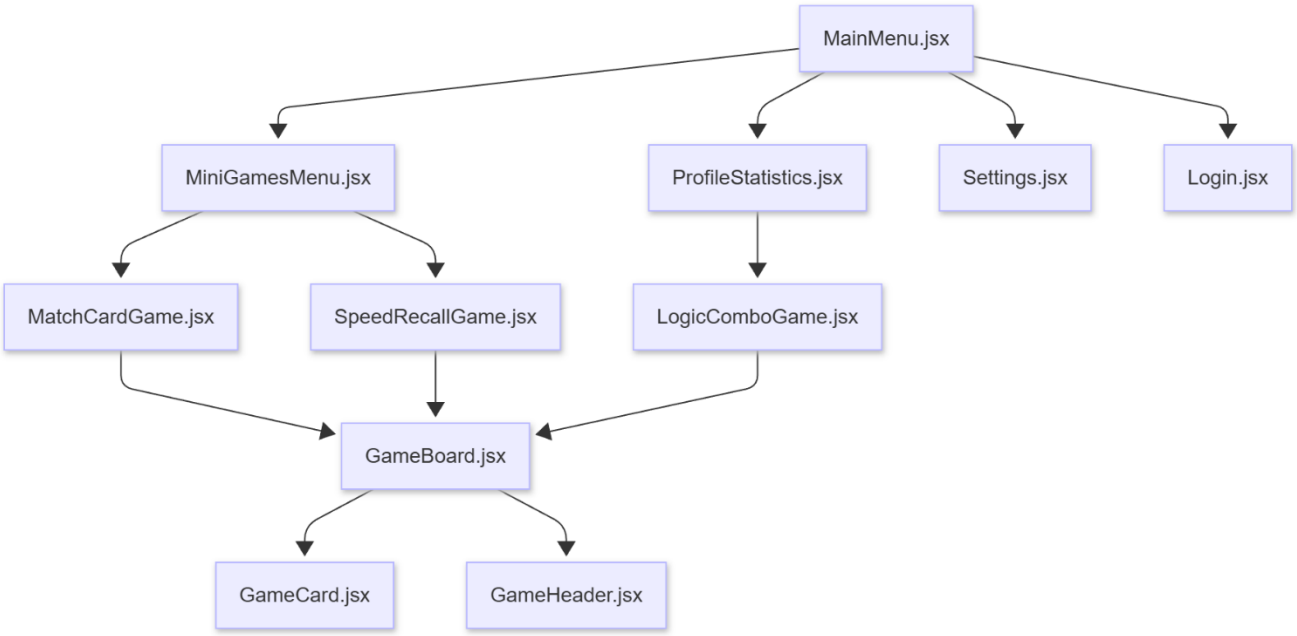


Рисунок 4.4 – Структура основних компонентів фронтенда системи Memory+

Головне меню реалізоване компонентом MainMenu.jsx і надає доступ до основних розділів системи: вибору ігрових режимів, аналітики профілю, налаштувань та авторизації. Саме з цього екрана користувач розпочинає взаємодію із застосунком, а тому він містить найбільш важливі навігаційні елементи. Далі користувач переходить до MiniGamesMenu.jsx, де відображаються доступні когнітивні режими: класичний пошук пар, швидке запам’ятовування та комбінований логічний режим.

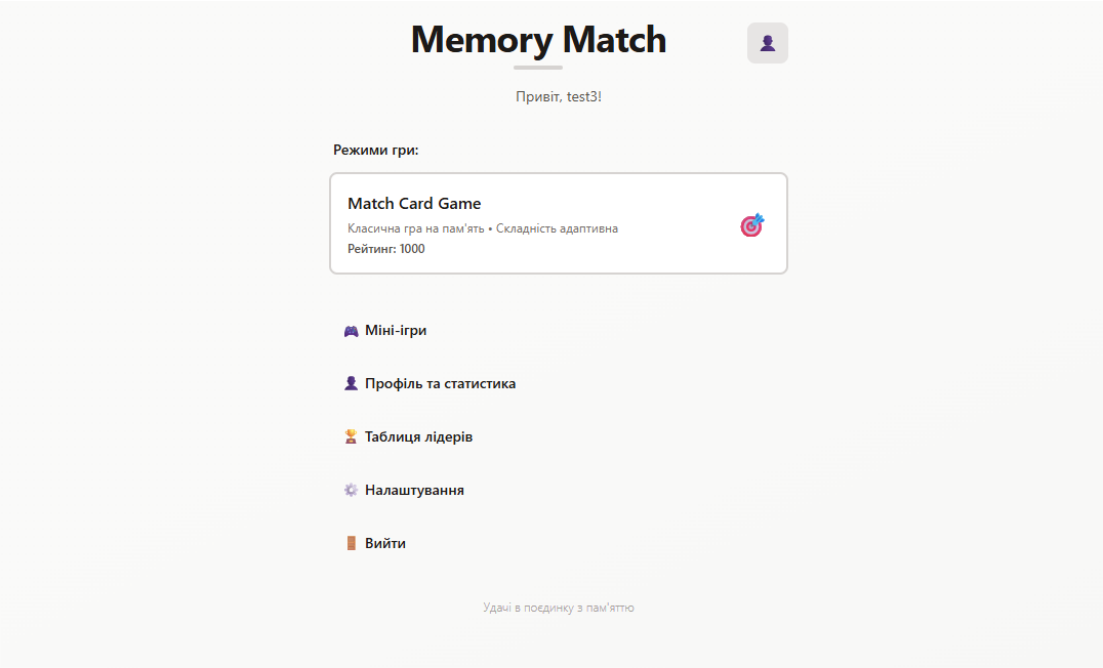


Рисунок 4.5 – Головне меню системи Memory+

Ігрові режими реалізовані трьома незалежними компонентами, кожен з яких має власну внутрішню логіку:

MatchCardGame.jsx - класична гра на пошук пар, орієнтована на розвиток короткочасної пам’яті та концентрації уваги.

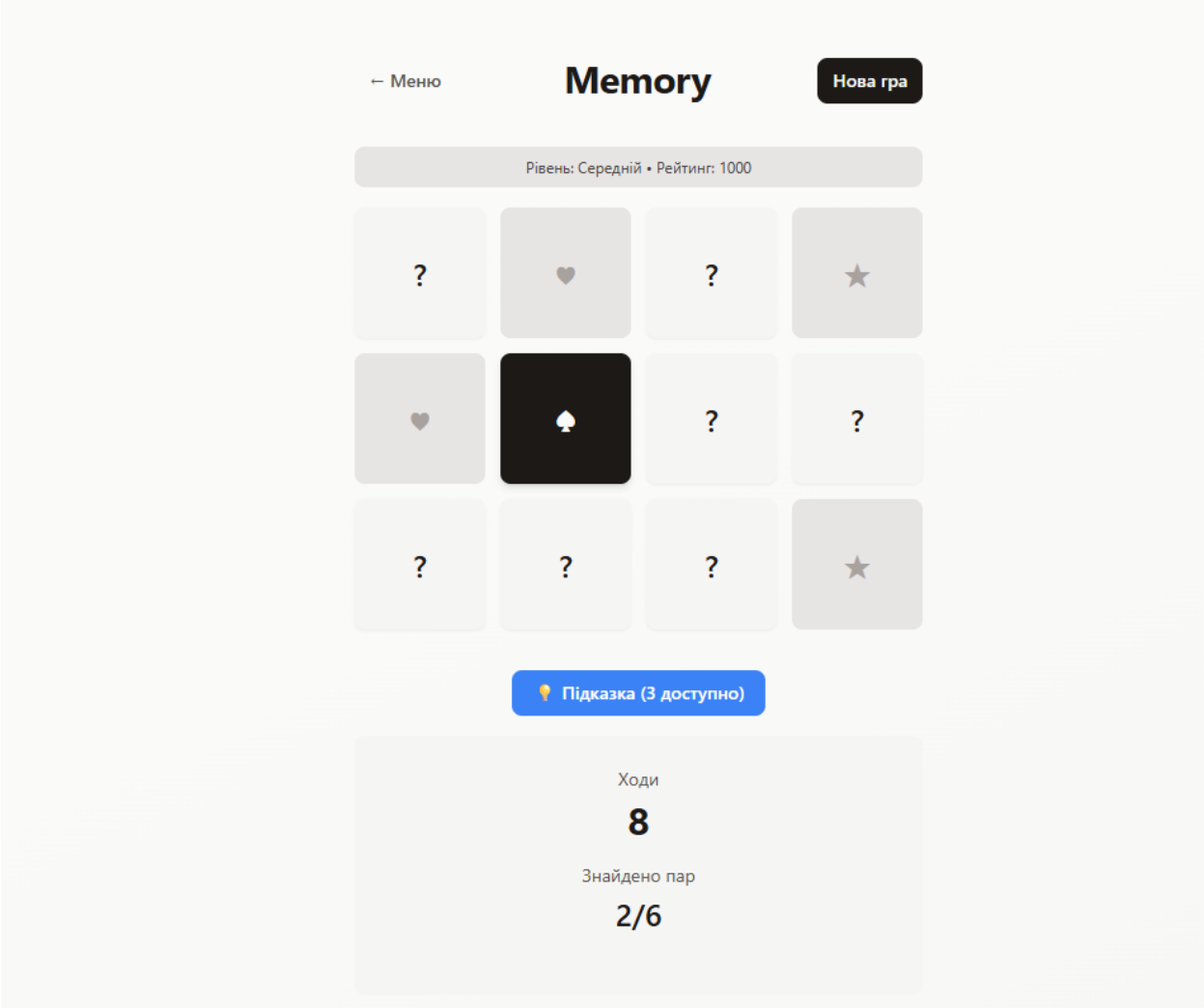


Рисунок 4.6 – Інтерфейс MatchCardGame

SpeedRecallGame.jsx — режим, що тренує швидкість запам’ятовування та відтворення інформації. Його робота складається з двох основних етапів: етапу демонстрації, коли користувач упродовж обмеженого часу переглядає набір елементів, та етапу відповіді, під час якого необхідно відтворити побачену послідовність або вибрати правильні елементи серед відволікаючих варіантів.

Механіка режиму дозволяє оцінювати здатність користувача до короткочасного запам’ятовування, швидкості реакції та точності прийняття рішень. Показники

виконання кожної спроби фіксуються та передаються до аналітичних модулів системи, що забезпечує подальшу адаптацію складності та персоналізоване навчання.

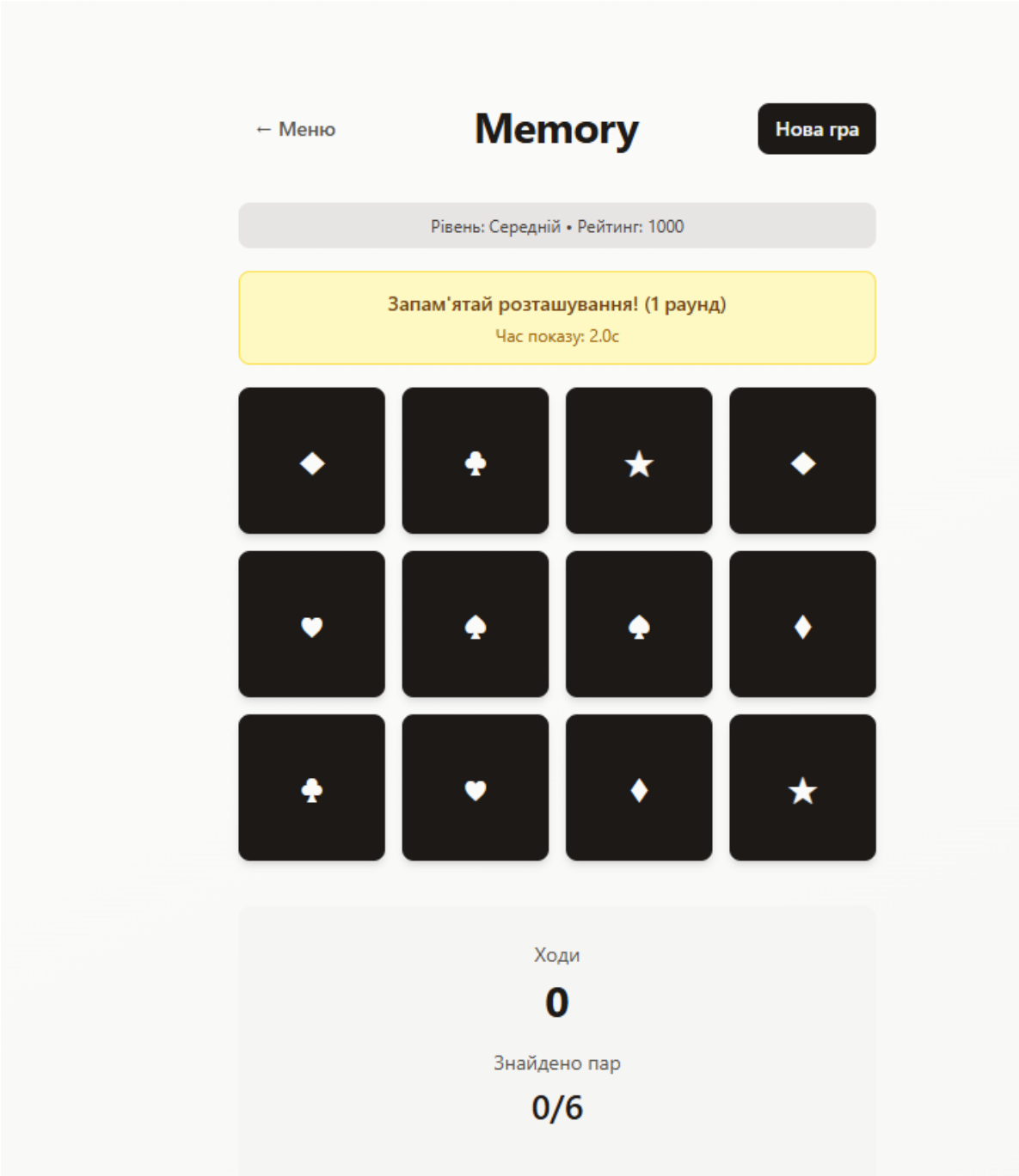


Рисунок 4.7 – Інтерфейс SpeedRecallGame

LogicComboGame.jsx - комплексний когнітивний тренажер, що поєднує завдання на пам’ять, логіку, швидкість мислення та аналіз послідовностей.

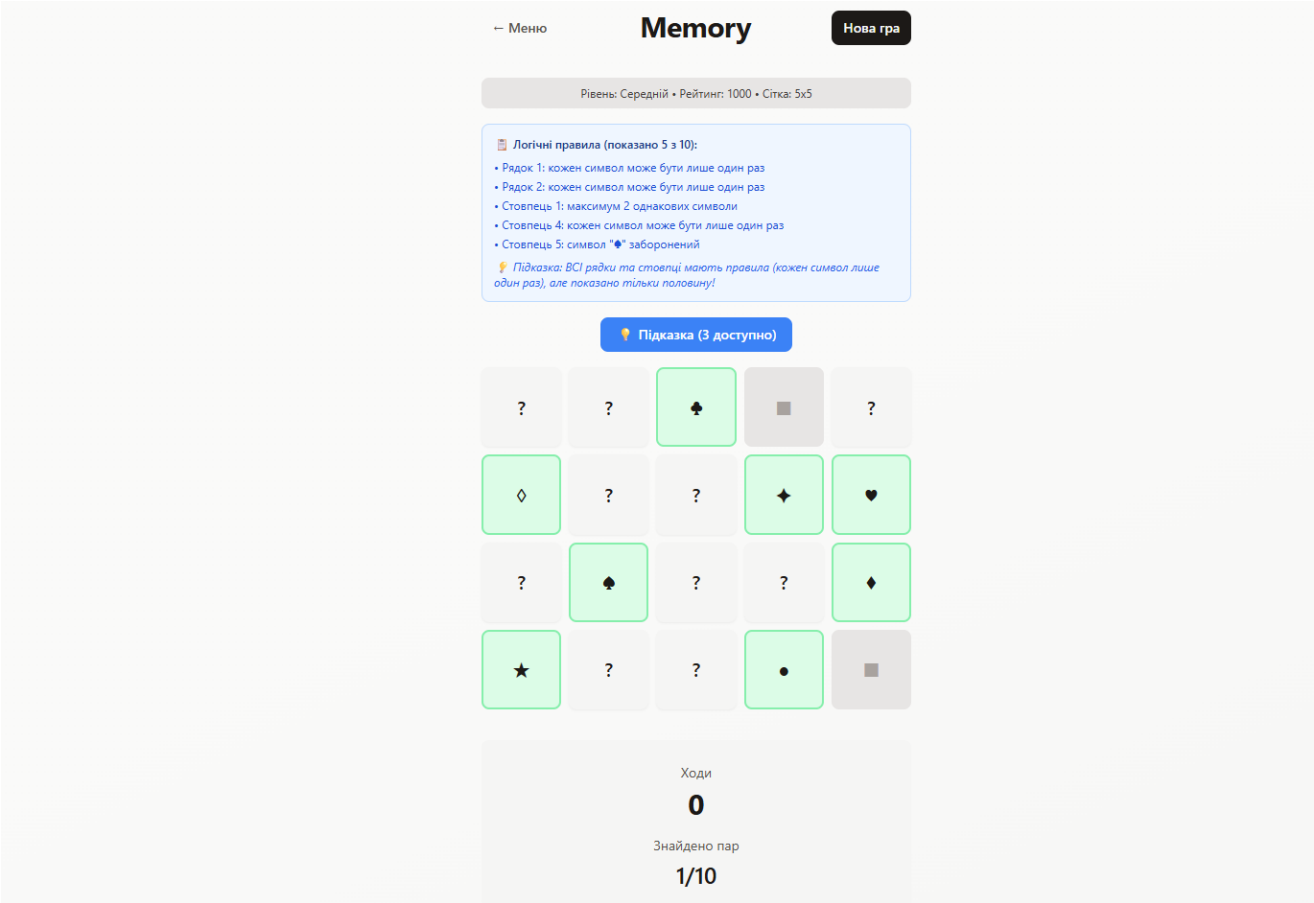


Рисунок 4.8 – Інтерфейс LogicComboGame

Усі ігрові режими використовують спільні низькорівневі компоненти інтерфейсу, такі як GameBoard.jsx, GameCard.jsx та GameHeader.jsx. Компонент GameBoard відповідає за формування структури ігрового поля, розташування карток та відображення їх станів, тоді як GameCard містить логіку перевороту, фіксації знайдених пар та обробку взаємодії користувача з окремою карткою. Компонент GameHeader відображає службові показники гри - час, кількість ходів, рівень складності та інші параметри.

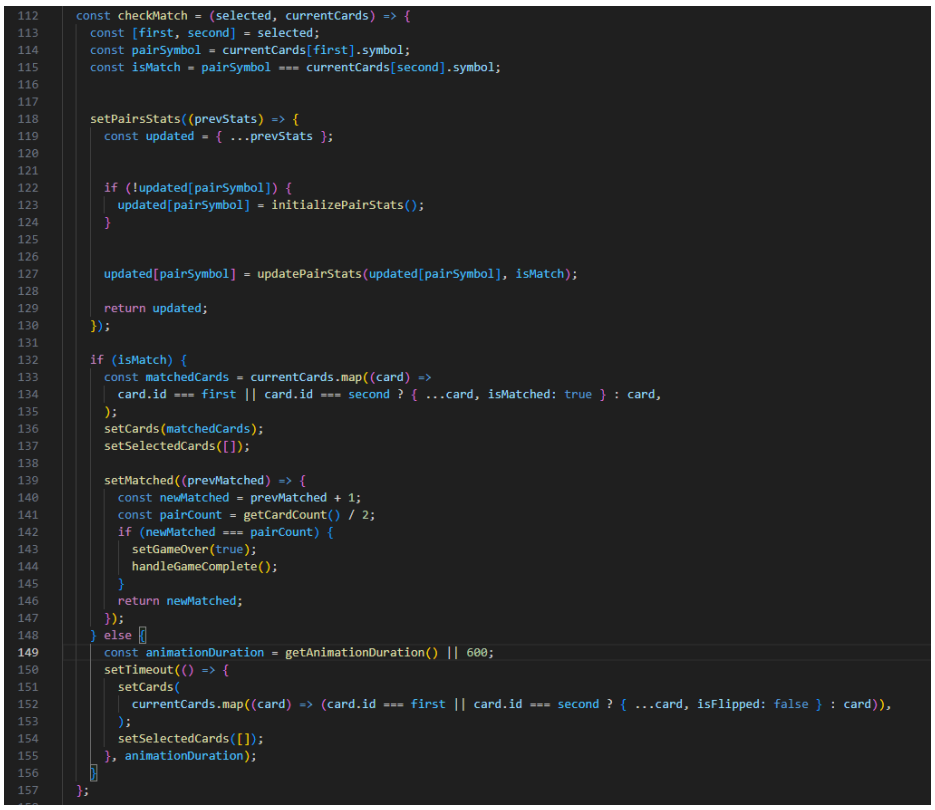
Поділ логіки на окремі функціональні блоки також полегшує тестування, оскільки кожен компонент можна перевіряти незалежно, що підвищує надійність та передбачуваність роботи всього застосунку

```jsx
import GameCard from "./GameCard";

function GameBoard({ cards, onCardClick, selectedCards }) {
  return (
    <div className="grid grid-cols-4 gap-3 mb-8">
      {cards.map((card) => (
        <GameCard
          key={card.id}
          card={card}
          isSelected={selectedCards.includes(card.id)}
          onClick={() => onCardClick(card.id)}
        />
      ))}
    </div>
  );
}

export default GameBoard;
```

Рисунок 4.9 – Фрагмент коду компонента GameBoard

```jsx
import { isAnimationsEnabled, getAnimationDuration } from "../utils/settings";

function GameCard({ card, isSelected, onClick }) {
  const animationsEnabled = isAnimationsEnabled();
  const animationDuration = getAnimationDuration();

  // Формуємо класи для анімації
  const transitionClass = animationsEnabled
    ? `transition-all`
    : '';
  const durationStyle = animationsEnabled
    ? { transitionDuration: `${animationDuration}ms` }
    : { transitionDuration: '0ms' };

  return (
    <button
      onClick={onClick}
      className={`aspect-square rounded-lg ${transitionClass} flex items-center justify-center text-2xl font-semibold cursor-pointer ${
        card.isMatched
          ? "bg-stone-200 text-stone-400 cursor-default"
          : card.isFlipped || isSelected
          ? "bg-stone-900 text-stone-50 shadow-md"
          : "bg-stone-100 text-stone-900 hover:bg-stone-200 shadow-sm"
      }`}
      style={durationStyle}
      disabled={card.isMatched}
    >
      {card.isFlipped || isSelected ? card.symbol : "?"}
    </button>
  );
}

export default GameCard;
```

Рисунок 4.10 – Фрагмент коду компонента GameCard



```
const checkMatch = (selected, currentCards) => {
  const [first, second] = selected;
  const pairSymbol = currentCards[first].symbol;
  const isMatch = pairSymbol === currentCards[second].symbol;


  setPairsStats((prevStats) => {
    const updated = { ...prevStats };


    if (!updated[pairSymbol]) {
      updated[pairSymbol] = initializePairStats();
    }


    updated[pairSymbol] = updatePairStats(updated[pairSymbol], isMatch);

    return updated;
  });

  if (isMatch) {
    const matchedCards = currentCards.map((card) =>
      card.id === first || card.id === second ? { ...card, isMatched: true } : card,
    );
    setCards(matchedCards);
    setSelectedCards([]);

    setMatched((prevMatched) => {
      const newMatched = prevMatched + 1;
      const pairCount = getCardCount() / 2;
      if (newMatched === pairCount) {
        setGameOver(true);
        handleGameComplete();
      }
      return newMatched;
    });
  } else {
    const animationDuration = getAnimationDuration() || 600;
    setTimeout(() => {
      setCards(
        currentCards.map((card) => (card.id === first || card.id === second ? { ...card, isFlipped: false } : card)),
      );
      setSelectedCards([]);
    }, animationDuration);
  }
};
```

Рисунок 4.11 – Логіка класичного режиму

```
setPairsStats((prevStats) => {
  const updated = { ...prevStats };


  if (!updated[pairSymbol]) {
    updated[pairSymbol] = initializePairStats();
  }


  updated[pairSymbol] = updatePairStats(updated[pairSymbol], isMatch);

  return updated;
});

if (isMatch) {
  const matchedCards = currentCards.map((card) =>
    card.id === first || card.id === second ? { ...card, isMatched: true } : card,
  );
  setCards(matchedCards);
  setSelectedCards([]);

  setMatched((prevMatched) => {
    const newMatched = prevMatched + 1;
    const pairCount = getCardCount() / 2;

    if (newMatched === pairCount) {
      setGameOver(true);
      handleGameComplete();
    } else {

      const newRevealTime = Math.max(500, revealTime - 150);
      setRevealTime(newRevealTime);
      setRound((prevRound) => prevRound + 1);
    }
    return newMatched;
  });
} else {
  const animationDuration = getAnimationDuration() || 600;
  setTimeout(() => {
    setCards(
      currentCards.map((card) => (card.id === first || card.id === second ? { ...card, isFlipped: false } : card)),
    );
    setSelectedCards([]);
  }, animationDuration);
}
};
```

Рисунок 4.12 – Логіка швидкісного режиму

```
const checkLogicRules = (firstIndex, secondIndex, symbol, currentCards, rulesToCheck = null) => {
  const rulesToUse = rulesToCheck || rules;
  const firstRow = Math.floor(firstIndex / gridSize);
  const firstCol = firstIndex % gridSize;
  const secondRow = Math.floor(secondIndex / gridSize);
  const secondCol = secondIndex % gridSize;


  const isActiveCard = (card) => card && (card.isMatched || card.isAlwaysVisible);


  const checkRuleInLine = (rule, lineIndices, symbolToCheck) => {
    if (!rule) return true;

    const activeCards = lineIndices
      .filter(idx => idx !== firstIndex && idx !== secondIndex)
      .map(idx => currentCards[idx])
      .filter(isActiveCard);

    switch (rule.ruleType) {
      case 'unique':

        return !activeCards.some(card => card.symbol === symbolToCheck);

      case 'forbidden_symbol':

        if (symbolToCheck === rule.forbiddenSymbol) {
          return false;
        }
        return true;

      case 'required_symbol':

        if (symbolToCheck === rule.requiredSymbol) {
          return true;
        }

        if (activeCards.some(card => card.symbol === rule.requiredSymbol)) {
          return true;
        }

        const totalCardsInLine = lineIndices.length;
        const remainingSlots = totalCardsInLine - activeCards.length - 2;
        if (remainingSlots === 0) {

          return symbolToCheck === rule.requiredSymbol;
        }
        return true;
```

Рисунок 4.13 – Логіка комбінованого режиму

Адаптивна логіка інтегрована у фронтенд через модулі microAnalysis.js, eloSystem.js та spacedRepetition.js. Під час кожної сесії гри компоненти збирають статистичну інформацію: кількість спроб, успішні відкриття, час реакції, послідовність дій користувача. Ці дані передаються в модуль мікроаналізу, який обчислює оновлені параметри ймовірності, ентропії, прогнозу та SM-2 інтервалів. Оновлені значення потім надсилаються на бекенд через маршрут /api/user/stats, де зберігаються в моделі користувача.

Статистичні та сервісні екрани користувача реалізовані компонентами ProfileStatistics.jsx, Settings.jsx та GameStats.jsx. Компонент Settings дозволяє налаштувати поведінку інтерфейсу - швидкість анімацій, режим підказок, параметри адаптивності. Компонент ProfileStatistics відображає рейтинг користувача, історію результатів, характеристики слабких і сильних пар, інтервальні повторення та загальну динаміку розвитку. Компонент GameStats.jsx надає деталізований аналіз окремих ігрових сесій, включаючи кількість помилок, час проходження, швидкість реакції, стабільність виконання дій та інші показники ефективності. Дані з цього модуля використовуються як для відображення статистики, так і для подальшої роботи адаптивних алгоритмів.

Завдяки такій структурі сервісні екрани виконують не лише роль інтерфейсних елементів, але й виступають інструментами зворотного зв’язку, які дозволяють користувачу усвідомити власні сильні сторони, прогалини та темп розвитку. Це також посилює мотивацію до регулярних тренувань і створює цілісну екосистему навчального прогресу.

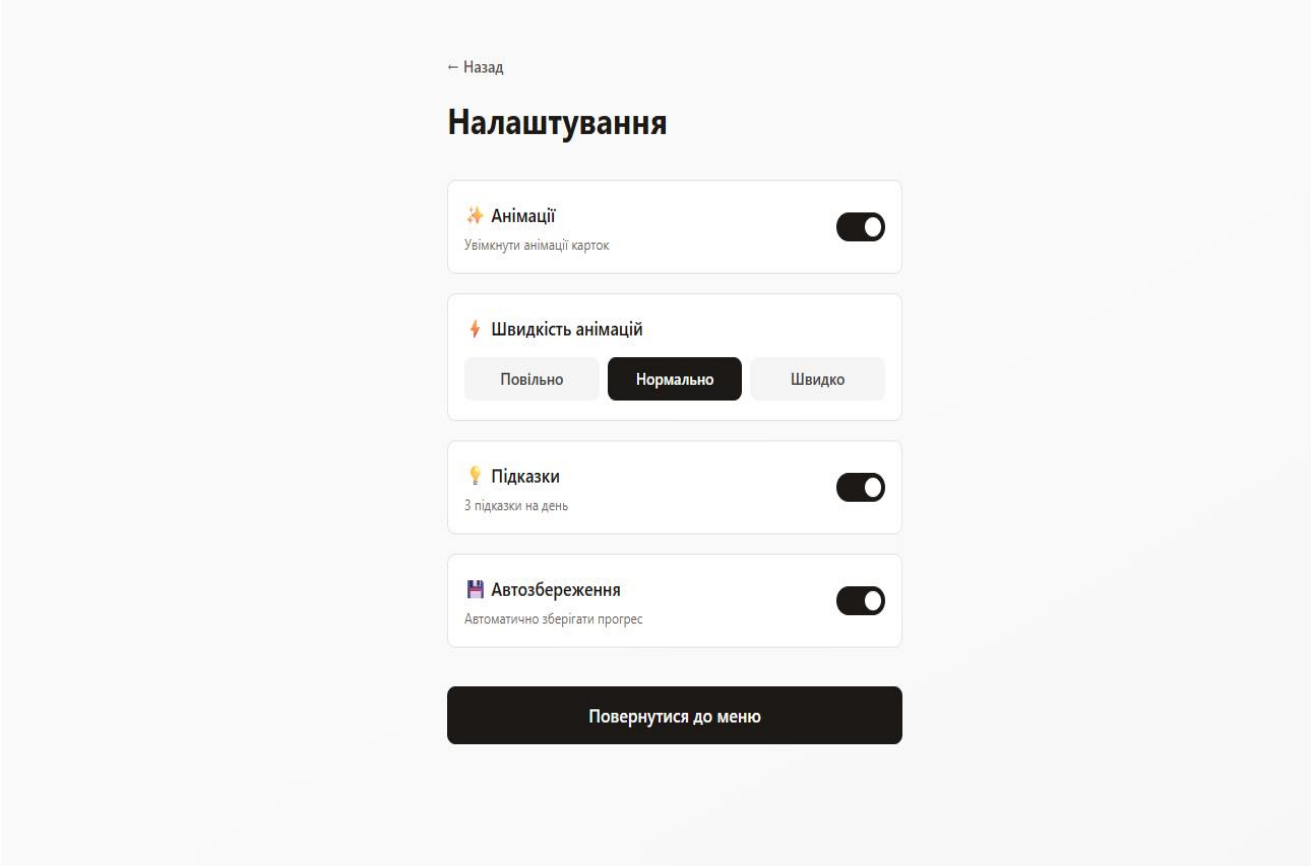


Рисунок 4.14 – Інтерфейс Settings.jsx

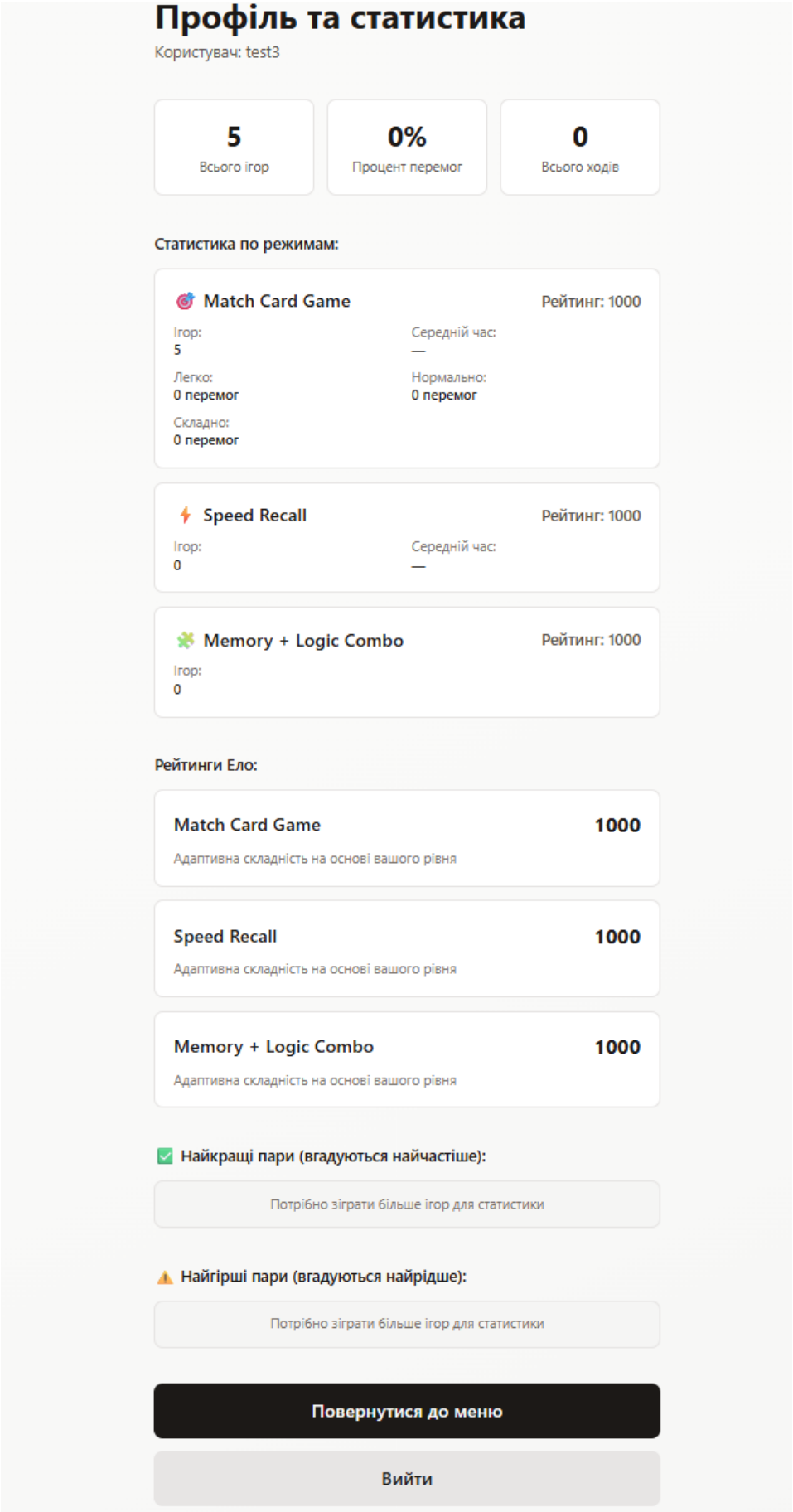


Рисунок 4.15 – Інтерфейс ProfileStatistics.jsx

Усі описані компоненти працюють у рамках спільного локального стану застосунку, забезпечуючи цілісність даних та узгоджену логіку переходів між режимами. Взаємодія фронтенда з бекендом реалізується через авторизовані HTTP-запити, що дозволяє синхронізувати результати ігрових сесій та підтримувати актуальний профіль користувача на будь-якому пристрої.

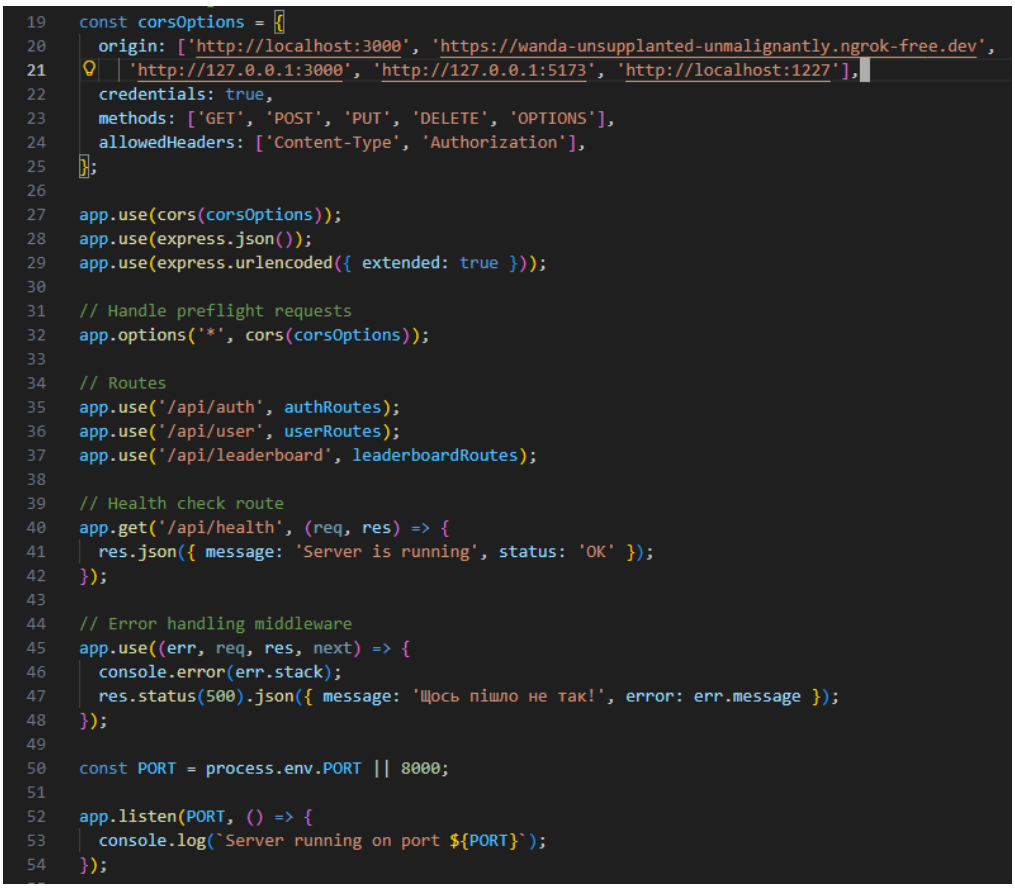
Завдяки гнучкій архітектурі на базі React фронтендна частина Memory+ забезпечує масштабованість, зручність використання та повну інтеграцію з адаптивними алгоритмами, що визначають унікальні навчальні траєкторії для кожного користувача.

4.3. Серверна частина системи

Серверна частина системи Memory+ реалізована на платформі Node.js із використанням фреймворку Express.js. Такий підхід забезпечує простоту розробки REST API, високу продуктивність під час обробки запитів та можливість масштабування у разі зростання навантаження. Бекенд системи відповідає за автентифікацію користувачів, приймання і збереження статистики, оновлення рейтингових параметрів, обробку даних адаптивних алгоритмів, формування таблиць лідерів та забезпечення взаємодії з базою даних MongoDB Atlas.

Архітектура бекенду структурована за класичним принципом розділення відповідальностей: окремо розміщено налаштування бази даних, middleware-функції, маршрути, контролери, моделі та конфігураційні файли. Це полегшує підтримку коду, підвищує читабельність і дозволяє без перешкод додавати нові модулі.

Основним файлом серверної частини є server.js, який виконує ініціалізацію сервера, налаштування middleware, підключення до бази даних та реєстрацію основних маршрутів. У ньому визначено роботу CORS-фільтра, JSON-парсера, а також основних REST-маршрутів /api/auth, /api/user та /api/leaderboard.



```
const corsOptions = {
  origin: ['http://localhost:3000', 'https://wanda-unsupplanted-unmalignantly.ngrok-free.dev',
    'http://127.0.0.1:3000', 'http://127.0.0.1:5173', 'http://localhost:1227'],
  credentials: true,
  methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
  allowedHeaders: ['Content-Type', 'Authorization'],
};

app.use(cors(corsOptions));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Handle preflight requests
app.options('*', cors(corsOptions));

// Routes
app.use('/api/auth', authRoutes);
app.use('/api/user', userRoutes);
app.use('/api/leaderboard', leaderboardRoutes);

// Health check route
app.get('/api/health', (req, res) => {
  res.json({ message: 'Server is running', status: 'OK' });
});

// Error handling middleware
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ message: 'Щось пішло не так!', error: err.message });
});

const PORT = process.env.PORT || 8000;

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

Рисунок 4.16 – Основний файл ініціалізації серверної частини

У серверній частині використано middleware-функцію auth.js, яка відповідає за перевірку автентичності користувачів. Для доступу до захищених маршрутів клієнт повинен надіслати токен у заголовку Authorization. Middleware декодує JWT-токен, отримує дані користувача та надає доступ до наступного обробника запиту.

```
import express from 'express';
import jwt from 'jsonwebtoken';
import User from '../models/User.js';

const router = express.Router();

const generateToken = (id) => {
  return jwt.sign({ id }, process.env.JWT_SECRET, {
    expiresIn: '30d',
  });
};

router.post('/register', async (req, res) => {
  try {
    const { username, password } = req.body;
```

Рисунок 4.17 – Перевірка JWT-токена у middleware auth.js

Обробку реєстрації та входу користувачів реалізовано у файлі маршрутизації auth.js. Сервер забезпечує два основні ендпоїнти:

- POST /api/auth/register – створення нового користувача;
- POST /api/auth/login – автентифікація та повернення JWT-токена.

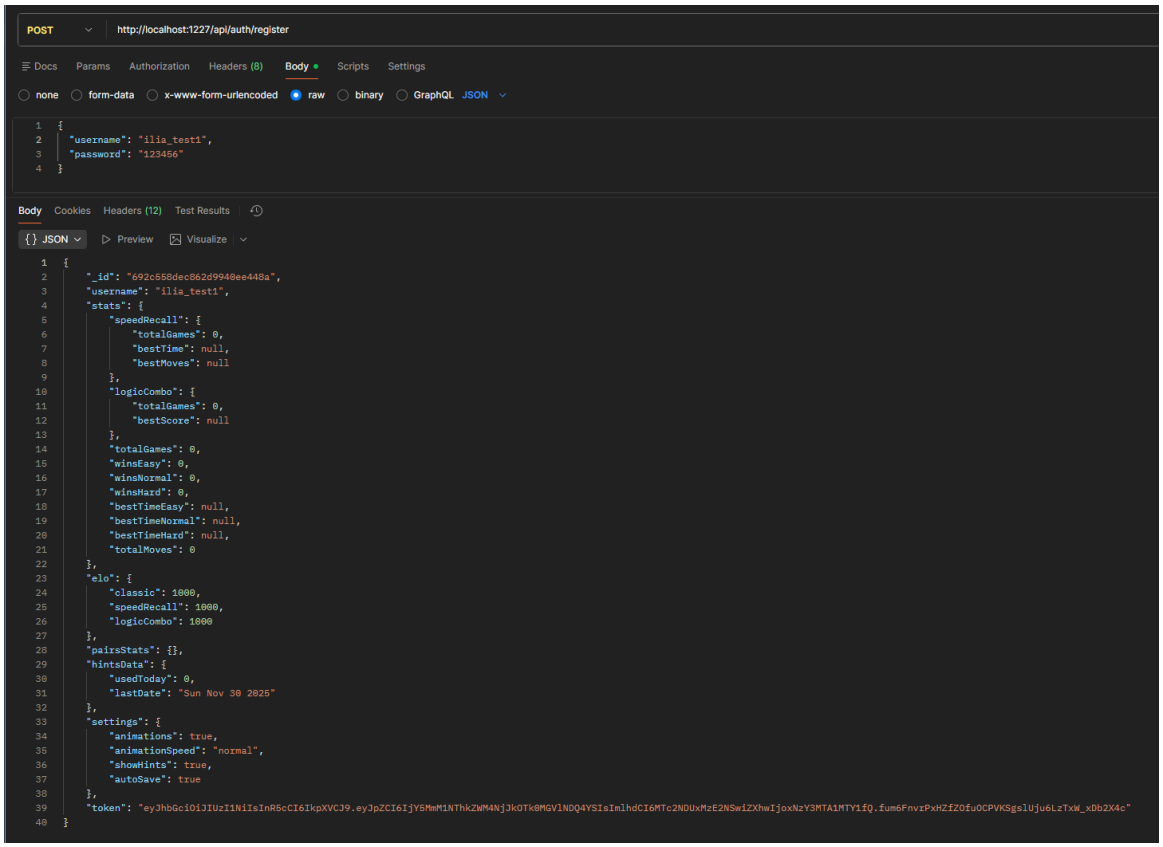
Під час реєстрації пароль хешується за допомогою бібліотеки bcryptjs, що гарантує захист користувацьких даних. Після успішної реєстрації або входу сервер повертає JSON-об’єкт із токеном та початковими параметрами профілю.

```
13  router.post('/register', async (req, res) => {
14    try {
15      const { username, password } = req.body;
16
17      if (!username || !password) {
18        return res.status(400).json({ message: 'Будь ласка, заповніть всі поля' });
19      }
20
21      if (username.length < 3) {
22        return res.status(400).json({ message: 'Ім\'я користувача повинно мати щонайменше 3 символи' });
23      }
24
25      const userExists = await User.findOne({ username });
26
27      if (userExists) {
28        return res.status(400).json({ message: 'Користувач з таким ім\'ям вже існує' });
29      }
30
31      console.log('👤 Creating new user:', username);
32      const user = await User.create({
33        username,
34        password,
35      });
36
37      if (user) {
38        console.log('✅ User created successfully in MongoDB:', user._id);
39        const userObj = user.toJSON();
40        res.status(201).json({
41          _id: userObj._id,
42          username: userObj.username,
43          stats: userObj.stats,
44          elo: userObj.elo,
45          pairsStats: userObj.pairsStats || {},
46          hintsData: userObj.hintsData,
47          settings: userObj.settings,
48          token: generateToken(user._id),
49        });
50      } else {
51        res.status(400).json({ message: 'Невдалося створити користувача' });
52      }
53    } catch (error) {
54      console.error(error);
55      res.status(500).json({ message: 'Помилка сервера' });
56    }
57  });
```

Рисунок 4.18 – Реалізація маршрутів POST /auth/register

```
59  router.post('/login', async (req, res) => {
60    try {
61      const { username, password } = req.body;
62
63
64      if (!username || !password) {
65        return res.status(400).json({ message: 'Будь ласка, заповніть всі поля' });
66      }
67
68      const user = await User.findOne({ username });
69
70      if (user && (await user.matchPassword(password))) {
71        const userObj = user.toJSON();
72        res.json({
73          _id: userObj._id,
74          username: userObj.username,
75          stats: userObj.stats,
76          elo: userObj.elo,
77          pairsStats: userObj.pairsStats || {},
78          hintsData: userObj.hintsData,
79          settings: userObj.settings,
80          token: generateToken(user._id),
81        });
82      } else {
83        res.status(401).json({ message: 'Невірне ім\'я користувача або пароль' });
84      }
85    } catch (error) {
86      console.error(error);
87      res.status(500).json({ message: 'Помилка сервера' });
88    }
89  });
```

Рисунок 4.19 – Реалізація маршрутів POST /auth/login



POST http://localhost:1227/api/auth/register

Docs Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
  "username": "ilia_test1",
  "password": "123456"
}
```

Body Cookies Headers (12) Test Results

{} JSON Preview Visualize

```
{
    "_id": "692c558dec862d9940ee448a",
    "username": "ilia_test1",
    "stats": {
        "speedRecall": {
            "totalGames": 0,
            "bestTime": null,
            "bestMoves": null
        },
        "logicCombo": {
            "totalGames": 0,
            "bestScore": null
        },
        "totalGames": 0,
        "winsEasy": 0,
        "winsNormal": 0,
        "winsHard": 0,
        "bestTimeEasy": null,
        "bestTimeNormal": null,
        "bestTimeHard": null,
        "totalMoves": 0
    },
    "elo": {
        "classic": 1000,
        "speedRecall": 1000,
        "logicCombo": 1000
    },
    "pairsStats": {},
    "hintsData": {
        "usedToday": 0,
        "lastDate": "Sun Nov 30 2025"
    },
    "settings": {
        "animations": true,
        "animationSpeed": "normal",
        "showHints": true,
        "autoSave": true
    },
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY5MmM1NThkZWM4NjJkOTk0MGVlNDQ4YSIsImlhdCI6MTc2NDUxMzE2NSwiZXhwIjoxNzY3MTA1MTY1fQ.fum6FnvzPxHZfZOfuOCPVKSgslUju6LzTxW_xDb2X4c"
}
```

Рисунок 4.20 – POST /auth/register у Postman

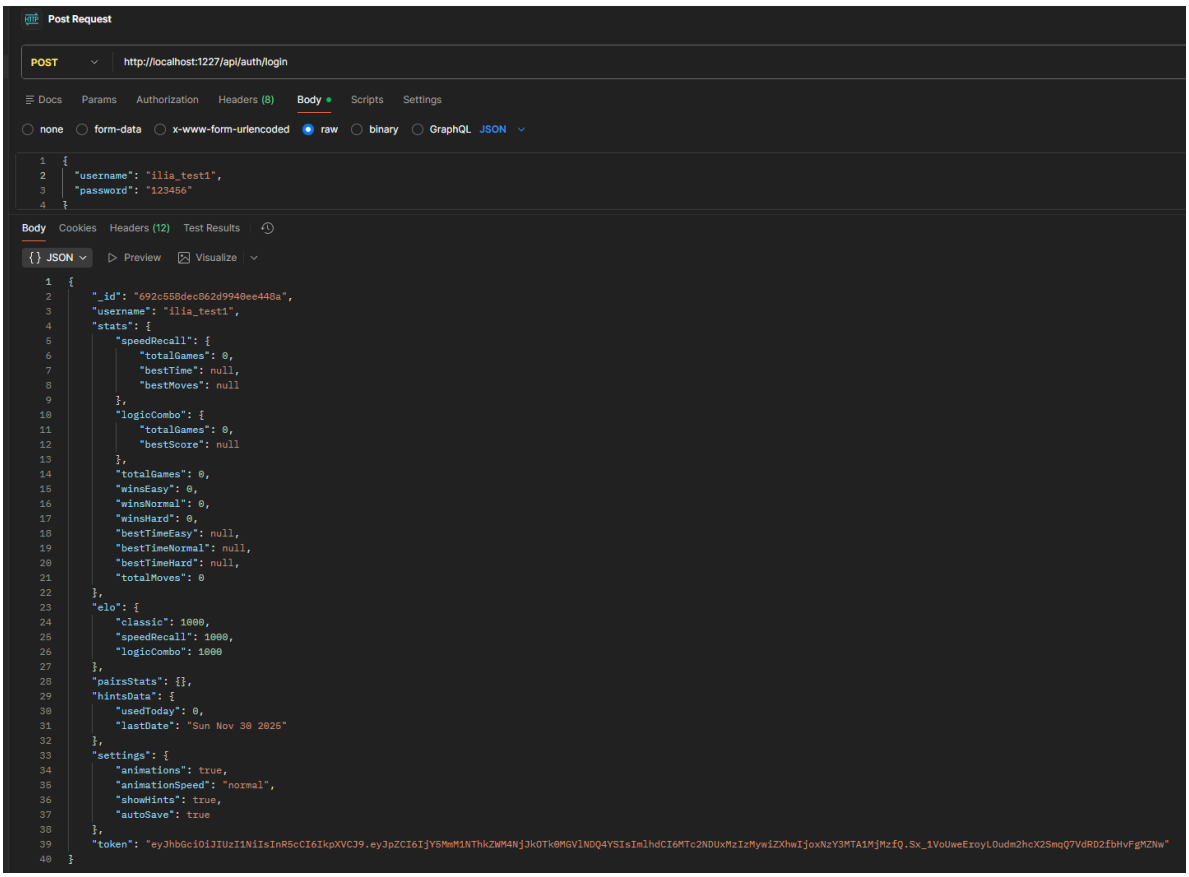


Рисунок 4.21 – POST /auth/login у Postman

Робота з профілем користувача, статистикою та адаптивними параметрами реалізована у маршрутах user.js. Система підтримує три основні операції:

1. GET /api/user/profile – повернення інформації про користувача.
2. POST /api/user/stats – приймання та збереження статистики сесії.
3. POST /api/user/settings – оновлення індивідуальних налаштувань.

Особливо важливим є маршрут /user/stats, який забезпечує інтеграцію з адаптивними алгоритмами. Фронтенд надсилає на сервер:

- оновлені значення рейтингу Elo;
- статистику режимів гри;
- дані мікроаналізу (ймовірності, ентропія, прогноз);
- параметри SM-2 (EF, repetition, interval);
- weakPairs і strongPairs;
- оновлені пари карток (pairsStats).

Сервер оновлює відповідні поля моделі User і зберігає результат у базі даних MongoDB Atlas.

```
router.put('/stats', protect, async (req, res) => {
  try {
    console.log('📊 Updating user stats for user:', req.user._id);
    console.log('📩 Received data:', JSON.stringify(req.body, null, 2));

    const updateData = {};


    if (req.body.stats) {
      Object.keys(req.body.stats).forEach(key => {
        if (key === 'speedRecall' || key === 'logicCombo') {

          Object.keys(req.body.stats[key]).forEach(nestedKey => {
            updateData[`stats.${key}.${nestedKey}`] = req.body.stats[key][nestedKey];
          });
        } else {
          updateData[`stats.${key}`] = req.body.stats[key];
        }
      });
    }

    if (req.body.elo) {
      Object.keys(req.body.elo).forEach(key => {
        updateData[`elo.${key}`] = req.body.elo[key];
      });
    }

    if (req.body.pairsStats) {
      const user = await User.findById(req.user._id);
      if (!user) {
        return res.status(404).json({ message: 'Користувача не знайдено' });
      }

```

Рисунок 4.22 – Обробка POST /user/stats у маршруті user.js

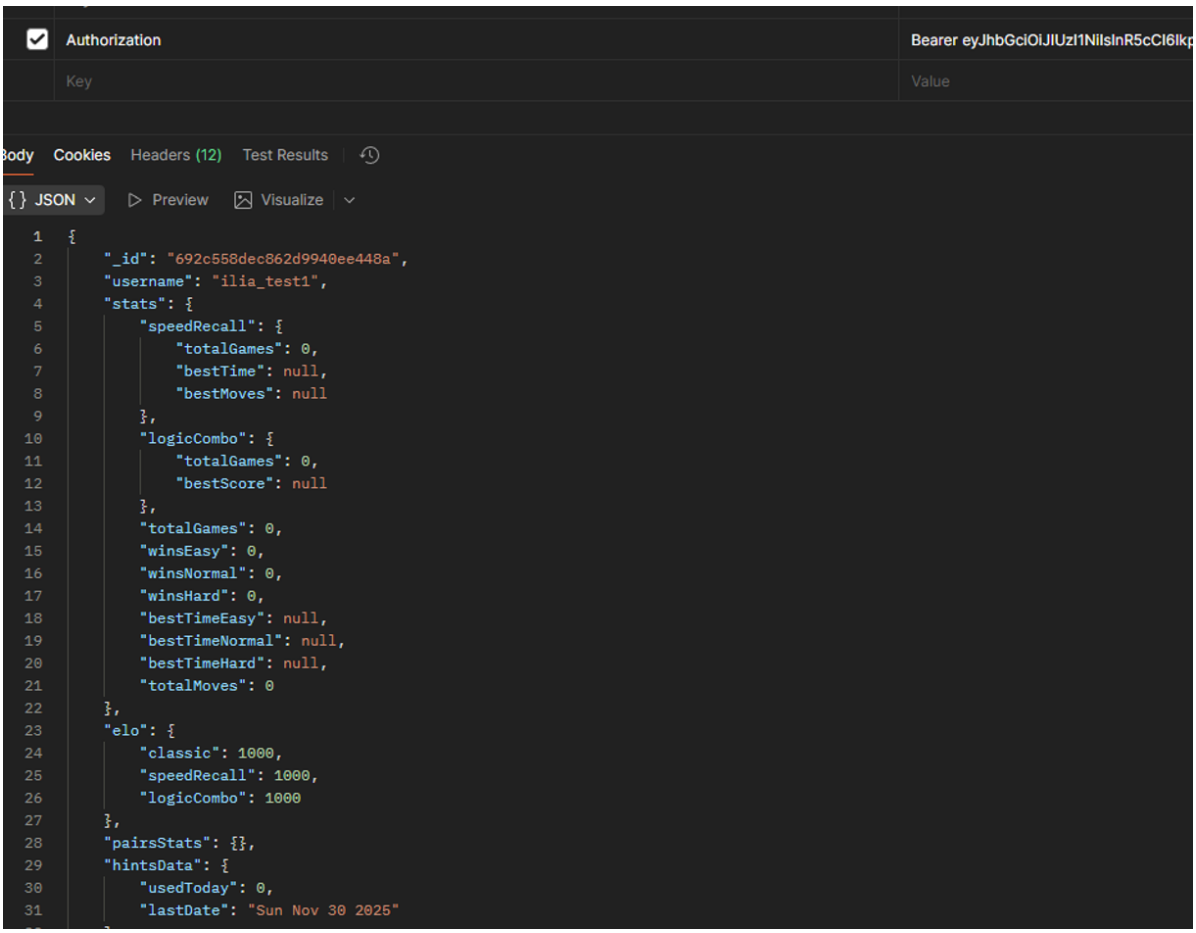


Рисунок 4.23 – GET /user/profile у Postman

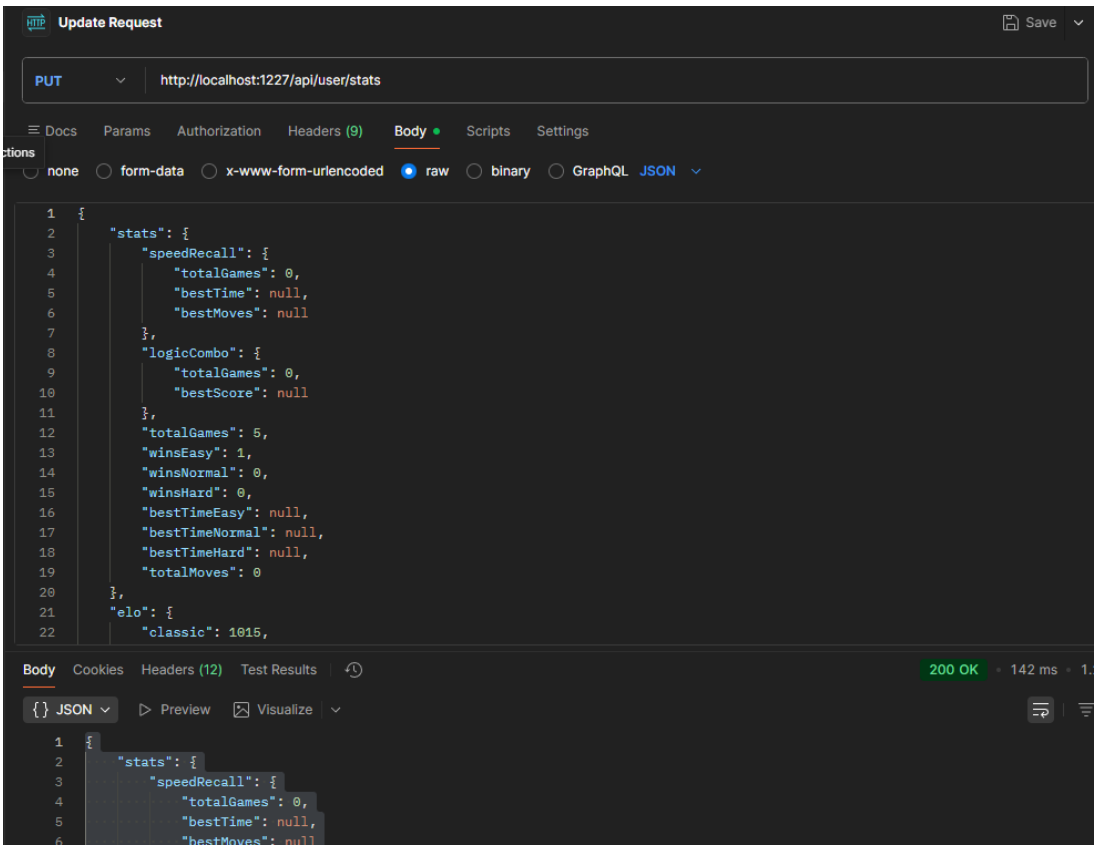


Рисунок 4.24 – PUT /user/stats у Postman

```
router.get('/', async (req, res) => {
  try {
    const { mode = 'classic', limit = 50 } = req.query;

    const validModes = ['classic', 'speedRecall', 'logicCombo'];
    if (!validModes.includes(mode)) {
      return res.status(400).json({ message: 'Невірний режим гри. Доступні: classic, speedRecall, logicCombo' });
    }

    const limitNum = parseInt(limit, 10);
    if (isNaN(limitNum) || limitNum < 1 || limitNum > 100) {
      return res.status(400).json({ message: 'Ліміт повинен бути від 1 до 100' });
    }

    const users = await User.find({})
      .select('username elo stats')
      .sort({ [`elo.${mode}`]: -1 })
      .limit(limitNum)
      .lean();

    const leaderboard = users.map((user, index) => ({
      rank: index + 1,
      username: user.username,
      elo: user.elo?.[mode] || 1000,
      stats: {
        totalGames: user.stats?.totalGames || 0,
        wins: mode === 'classic'
          ? (user.stats?.winsEasy || 0) + (user.stats?.winsNormal || 0) + (user.stats?.winsHard || 0)
          : mode === 'speedRecall'
          ? user.stats?.speedRecall?.totalGames || 0
          : user.stats?.logicCombo?.totalGames || 0,
        bestTime: mode === 'classic'
          ? user.stats?.bestTimeNormal || user.stats?.bestTimeEasy || user.stats?.bestTimeHard || null
          : mode === 'speedRecall'
          ? user.stats?.speedRecall?.bestTime || null
          : null,
        bestScore: mode === 'logicCombo'
          ? user.stats?.logicCombo?.bestScore || null
          : null,
      },
    }));
```

Рисунок 4.25 – Реалізація таблиці лідерів

Для визначення позицій у рейтинговій таблиці використовується маршрут leaderboard.js. Він забезпечує можливість отримати:

- повний рейтинг за режимом (/leaderboard?mode=classic),
- рейтинг навколо поточного користувача (/leaderboard/around).

Результати сортуються за відповідним рейтингом Elo.

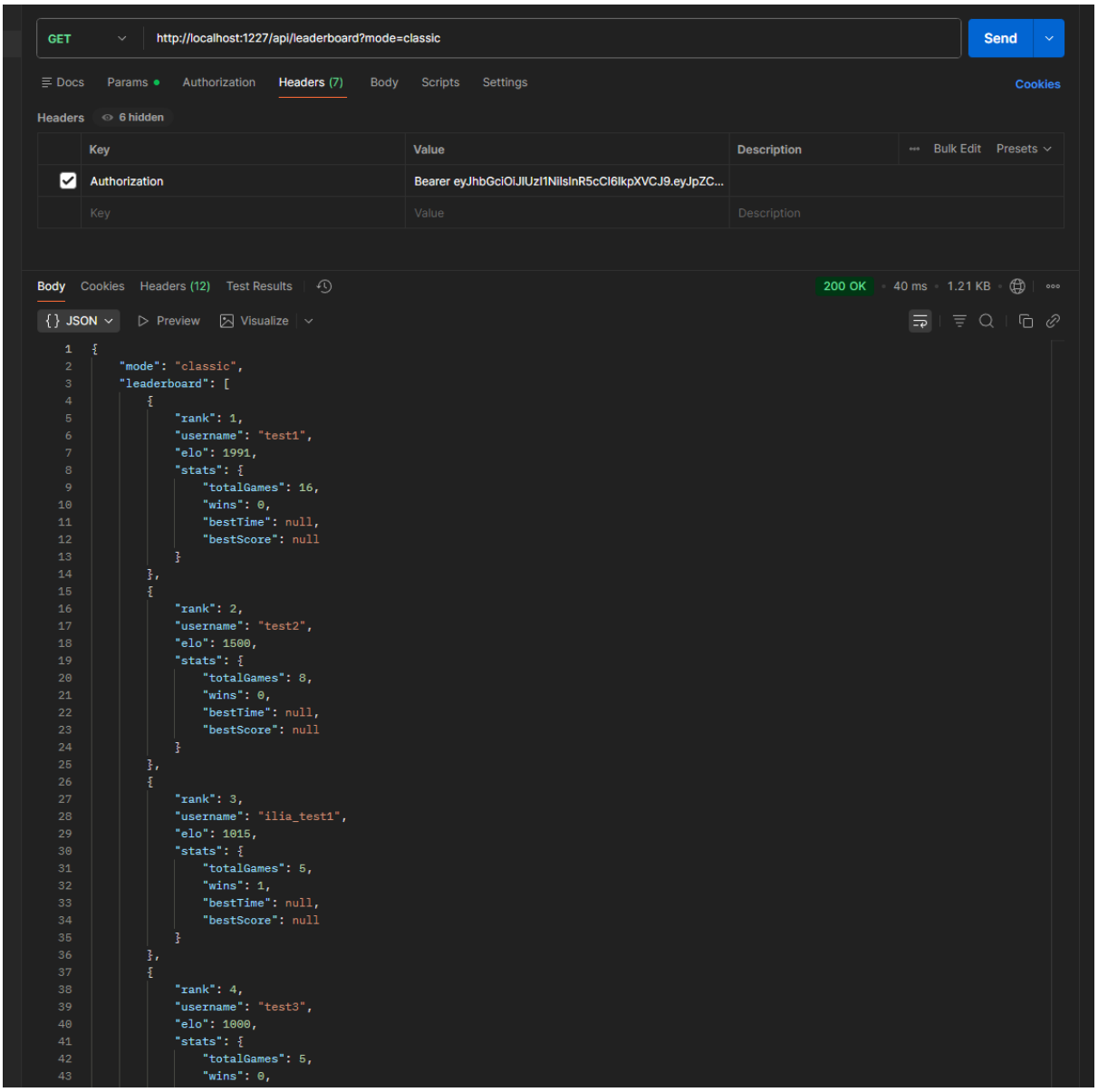


Рисунок 4.26 – GET /leaderboard у Postman

Безпека серверної частини забезпечується комплексно:

- JWT-токени використовуються для авторизації доступу до приватних ресурсів.
- bcryptjs шифрує паролі перед збереженням.

- CORS-політика обмежує дозволені джерела запитів.
- Помилки обробляються централізовано через middleware у server.js, що забезпечує стабільність роботи системи.

Серверна частина Memory+ створює надійний фундамент для аналітики та інтеграції адаптивних алгоритмів, забезпечує стабільний обмін даними з клієнтською частиною та гарантує коректне збереження всієї статистичної інформації.

4.4. База даних

База даних системи Memory+ побудована на основі документоорієнтованої СКБД MongoDB, що відзначається гнучкою структурою даних та можливістю масштабування. Для взаємодії з базою застосовано ORM-бібліотеку Mongoose, яка дозволяє визначати схеми документів, виконувати валідацію та створювати моделі.

Основним документом у базі даних є User, який містить повну інформацію про профіль користувача, його ігрову активність, прогрес у розвитку пам’яті, рейтинги Elo, статистику режимів гри, внутрішні параметри адаптивних моделей та дані алгоритму SM-2. Структура схеми побудована таким чином, щоб забезпечити можливість швидкої обробки статистичних даних, регулярного оновлення результатів та гнучкої інтеграції алгоритмів адаптації.

Окрім основного документа, система містить допоміжні структури, такі як GameSession, CardPair, EntropyMetrics, які дозволяють фіксувати перебіг кожної ігрової спроби та формувати повноцінний набір аналітичних характеристик. Завдяки цьому база даних підтримує як короткостроковий аналіз продуктивності користувача, так і формування довгострокового профілю когнітивного розвитку.

Документоорієнтована модель MongoDB забезпечує гнучкість у зберіганні вкладених структур і масивів подій, що є критично важливим для системи з високою частотою оновлень та потребою зберігати історію мікродій користувача. Такий підхід підвищує продуктивність запитів і дозволяє масштабувати систему зі збільшенням кількості користувачів без необхідності значних змін у структурі бази.

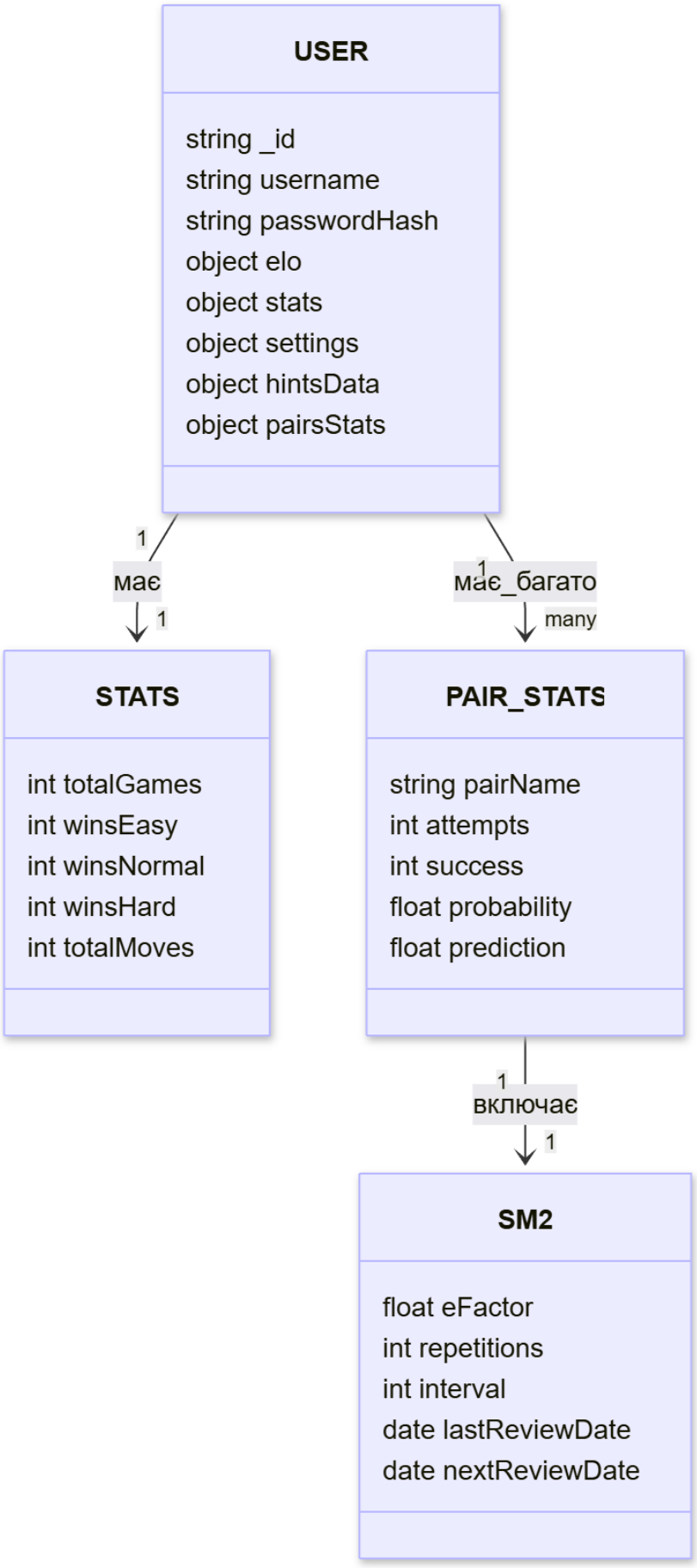


Рисунок 4.27 – ER-діаграма бази даних Memory+

Структура документа User

Документ User складається з кількох логічних блоків:

- особиста інформація користувача: _id, username, passwordHash;
- налаштування інтерфейсу та гри: об’єкт settings;
- рейтингова система: об’єкт elo для всіх режимів гри;
- статистика режимів: об’єкт stats;
- аналітичні дані: pairsStats, hintsData;
- системні метадані: createdAt, updatedAt.

Ця структура дозволяє застосовувати до кожного користувача індивідуальні адаптивні моделі, включно з алгоритмом Elo, microAnalysis і системою інтервальних повторень SM-2.

```
5   const router = express.Router();
6
7   router.get('/profile', protect, async (req, res) => {
8     try {
9       const user = await User.findById(req.user._id);
10      if (!user) {
11        return res.status(404).json({ message: 'Користувача не знайдено' });
12      }
13
14      const userObj = user.toJSON();
15      res.json({
16        _id: userObj._id,
17        username: userObj.username,
18        stats: userObj.stats,
19        elo: userObj.elo,
20        pairsStats: userObj.pairsStats || {},
21        hintsData: userObj.hintsData,
22        settings: userObj.settings,
23      });
24    } catch (error) {
25      console.error(error);
26      res.status(500).json({ message: 'Помилка сервера' });
27    }
28  });
```

Рисунок 4.28 – Структура документа User у базі даних

Статистичні структури: stats і pairsStats

1. Блок stats

Елемент stats містить кількісні показники продуктивності користувача в різних ігрових режимах. Він використовується для відстеження прогресу та формування динаміки розвитку:

- кількість зіграних ігор;
- найкращий результат (час/хід/бал);
- перемоги на рівнях Easy/Normal/Hard;
- дані ігрових режимів SpeedRecall і LogicCombo;
- загальні метрики (totalGames, totalMoves).

Бекенд може оновлювати ці дані після кожної сесії (PUT /user/stats).

2. Блок pairsStats

Цей блок є центральним елементом адаптивної логіки системи. Він містить детальну статистику для кожної карткової пари (наприклад, sun, moon, cloud, tree) та використовується для динамічного коригування складності гри. Кожен елемент структури описує рівень засвоєння конкретної пари та забезпечує умови для подальшого персоналізованого навчання.

До параметрів, що зберігаються в pairsStats, належать:

• attempts – загальна кількість спроб користувача взаємодії з даною парою;

• success – кількість успішних відкриттів;

• probability – емпірична ймовірність правильної відповіді (success/attempts);

• prediction – прогноз наступного успіху, який обчислюється методом експоненціального згладжування;

• sm2 – внутрішній стан алгоритму інтервальних повторень SM-2, що включає:

  • eFactor – коефіцієнт легкості запам’ятовування;

  • repetitions – кількість повторів пари;

  • interval – розрахований інтервал повторення у днях;

  • nextReviewDate – дата наступного появлення пари;

  • lastReviewDate – дата останнього повторення.

Завдяки такій структурі pairsStats виступає основним джерелом інформації для алгоритмів адаптивності. Саме на основі цих даних система визначає, які картки потребують більш частих повторень, а які вже добре засвоєні користувачем. Таким чином слабкі пари демонструються частіше, а сильні — рідше.

```
107     if (req.body.pairsStats) {
108       const user = await User.findById(req.user._id);
109       if (!user) {
110         return res.status(404).json({ message: 'Користувача не знайдено' });
111       }
112
113
114       Object.keys(req.body.pairsStats).forEach(symbol => {
115         user.pairsStats.set(symbol, req.body.pairsStats[symbol]);
116       });
117       await user.save();
118     }
119
120     if (req.body.hintsData) {
121       updateData['hintsData'] = req.body.hintsData;
122     }
123
124     if (Object.keys(updateData).length > 0) {
125       console.log('💾 Saving to MongoDB with updateData:', JSON.stringify(updateData, null, 2));
126
127       const user = await User.findByIdAndUpdate(
128         req.user._id,
129         { $set: updateData },
130         { new: true, runValidators: true }
131       );
132
133       if (!user) {
134         console.error('❌ User not found:', req.user._id);
135         return res.status(404).json({ message: 'Користувача не знайдено' });
136       }
137
138       console.log('✅ User stats updated successfully');
139       const userObj = user.toJSON();
140       res.json({
141         stats: userObj.stats,
142         elo: userObj.elo,
143         pairsStats: userObj.pairsStats || {},
144         hintsData: userObj.hintsData,
145       });
146     } else if (req.body.pairsStats) {
147
148       const user = await User.findById(req.user._id);
149       const userObj = user.toJSON();
150       res.json({
151         pairsStats: userObj.pairsStats || {},
152       });
153     } else {
154       res.status(400).json({ message: 'Немає даних для оновлення' });
```

Рисунок 4.29 – Структура statistics / pairsStats у документі User

4.5. Інтеграція адаптивних алгоритмів у код

Адаптивність є ключовою властивістю системи Memory+, оскільки вона дозволяє підлаштовувати складність гри, інтенсивність повторень, структуру карткових пар та індивідуальну траєкторію навчання під кожного користувача. Інтеграція адаптивних алгоритмів реалізована на фронтенді у вигляді окремих модулів (microAnalysis.js, spacedRepetition.js, eloSystem.js) та у серверній частині через збереження результатів у моделі користувача (User.js). Зв’язок між

компонентами здійснюється через API-метод /api/user/stats, який приймає результати ігрової сесії та оновлює відповідні дані у базі даних.

Після кожної ігрової сесії фронтенд обчислює статистичні метрики, оновлює рейтинги, перераховує параметри SM-2 та формує оновлену структуру pairsStats. Отримані значення передаються на сервер у вигляді JSON-об’єкта. Бекенд виконує валідацію, збереження та формування нової адаптивної моделі навчання для наступної гри.

Одним з найважливіших компонентів адаптивності є модуль microAnalysis.js, який обчислює локальні показники поведінки користувача: ймовірність успіху, прогнозовану успішність за допомогою експоненціального згладжування, а також ентропію Шеннона як характеристику складності. Ці значення використовуються у подальшому для формування слабких та сильних карткових пар і визначення очікуваної результативності.

```
79
80  export function calculateEntropy(pairsStats) {
81    if (!pairsStats || Object.keys(pairsStats).length === 0) {
82      return 0;
83    }
84
85    let entropy = 0;
86    const pairs = Object.values(pairsStats);
87
88    for (const pair of pairs) {
89      const p = pair.probability;
90
91
92      if (p > 0 && p < 1) {
93        entropy -= p * Math.log2(p);
94      }
95    }
96
97    return entropy;
98  }
99
100 export function normalizeEntropy(entropy, pairCount) {
101   if (pairCount <= 1) return 0;
102
103
104   const maxEntropy = Math.log2(pairCount);
105   return maxEntropy > 0 ? Math.min(1, entropy / maxEntropy) : 0;
106 }
107
```

Рисунок 4.30 – Фрагмент microAnalysis.js

Другим модулем є spacedRepetition.js, який реалізує алгоритм інтервального повторення SM-2. Він визначає, наскільки добре користувач запам’ятовує конкретну карткову пару, і коригує коефіцієнт легкості (EF), кількість повторень та інтервал до наступного показу. Отримані параметри зберігаються у полі pairsStats кожної пари.

```
120     sm2.eFactor = updateEFactor(sm2.eFactor, quality);
121
122
123     if (wasSuccessful && quality >= 3) {
124       sm2.repetition += 1;
125       sm2.interval = calculateInterval(sm2.repetition, sm2.eFactor);
126       sm2.nextReviewDate = calculateNextReviewDate(currentDate, sm2.interval).toISOString();
127     } else {
128
129       sm2.repetition = 0;
130       sm2.interval = 1;
131       sm2.nextReviewDate = calculateNextReviewDate(currentDate, 1).toISOString();
132     }
133
134     sm2.lastReviewDate = currentDate.toISOString();
135     sm2.totalReviews += 1;
136
137     return updated;
138   }
139
140
141   export function getCardsForReview(pairsStats, currentDate = new Date()) {
142     const reviewCards = [];
143
144     for (const [symbol, stats] of Object.entries(pairsStats)) {
145       if (shouldReviewNow(stats, currentDate)) {
146
147         const priority = stats.sm2?.nextReviewDate
148           ? new Date(currentDate) - new Date(stats.sm2.nextReviewDate)
149           : Infinity;
150
151         reviewCards.push({
152           symbol,
153           stats,
154           priority,
155         });
156       }
157     }
```

Рисунок 4.31 – Фрагмент spacedRepetition.js

Алгоритм рейтингу Elo, реалізований у модулі eloSystem.js, забезпечує автоматичне регулювання складності. Після кожної сесії обчислюється очікувана результативність гравця та новий рейтинг, який впливає на кількість карткових пар, дозволені підказки та швидкість появи сильних/слабких карт. Система Elo використовується окремо для кожного режиму, що дозволяє точно оцінювати здібності гравця в різних типах задач.

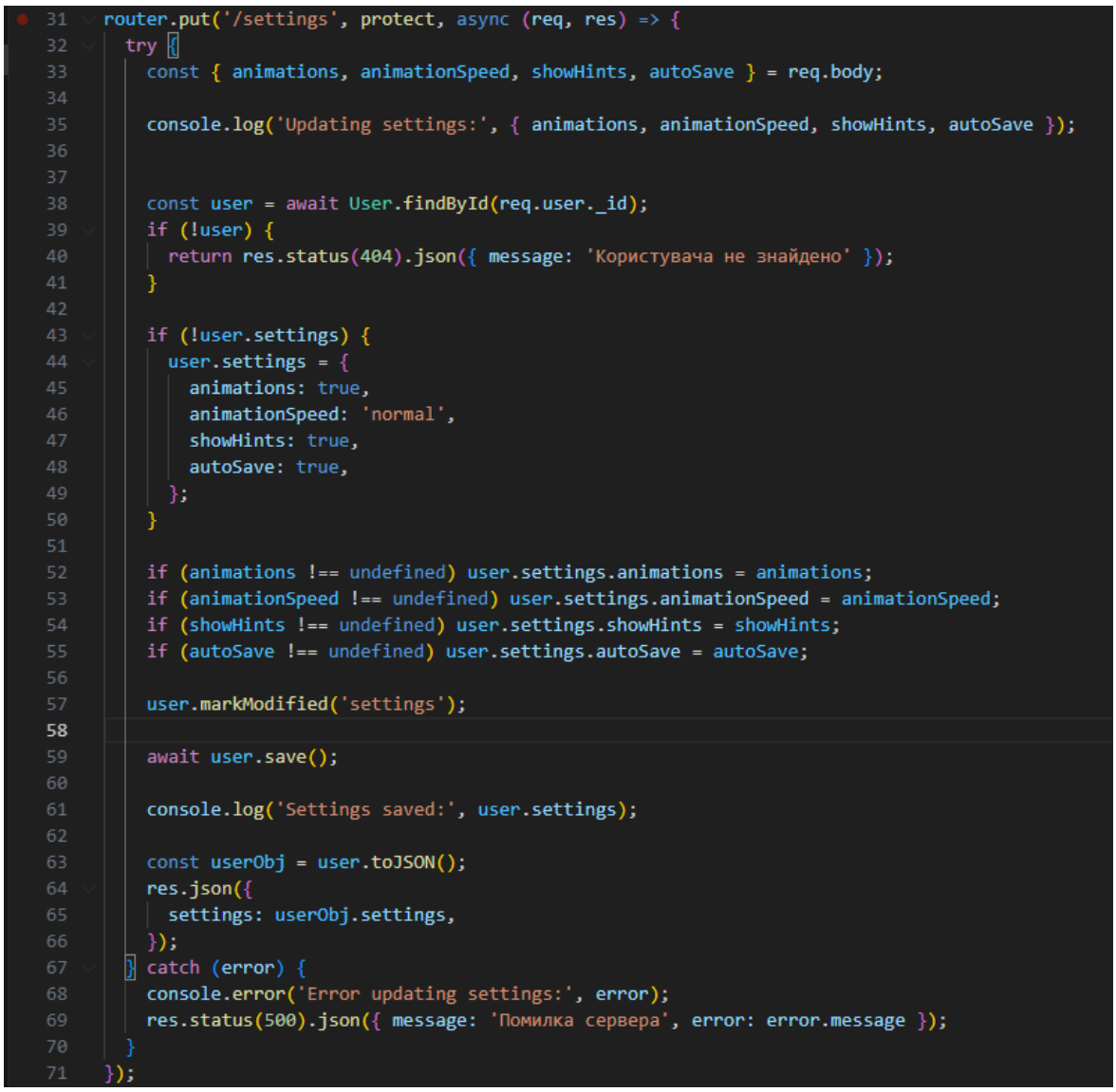
```
33
34  export function calculateActualScore(moves, matched, totalPairs, timeSpent, gameType) {
35
36    if (gameType === 'speedRecall') {
37
38      const optimalMoves = totalPairs;
39      const moveEfficiency = Math.max(0, 1 - (moves - optimalMoves) / (totalPairs * 2));
40
41
42      const optimalTime = totalPairs * 2;
43      const timeEfficiency = Math.max(0, 1 - (timeSpent - optimalTime) / (optimalTime * 2));
44
45      return (moveEfficiency * 0.6 + timeEfficiency * 0.4);
46    } else {
47
48      const optimalMoves = totalPairs;
49      const moveEfficiency = Math.max(0, 1 - (moves - optimalMoves) / (totalPairs * 3));
50
51      const optimalTime = totalPairs * 5;
52      const timeEfficiency = Math.max(0, 1 - (timeSpent - optimalTime) / (optimalTime * 2));
53
54      return (moveEfficiency * 0.8 + timeEfficiency * 0.2);
55    }
56  }
57
```

Рисунок 4.32 – Оновлення рейтингу в eloSystem.js

Збереження аналітичних даних виконується у моделі User.js. Поле pairsStats містить інформацію про кожну карткову пару: кількість спроб, прогнозовану успішність, ймовірність, а також параметри SM-2 (EF, interval, repetitions, nextReviewDate, lastReviewDate). Значення оновлюються при кожному виклику маршруту PUT /api/user/stats. Така структура дозволяє зберігати довготривалу історію тренувань і створювати персоналізовану модель когнітивного розвитку.

Крім того, такий формат організації даних забезпечує можливість формування адаптивних навчальних сценаріїв, оскільки кожен елемент статистики може бути використаний для коригування складності наступних ігор. Система отримує змогу аналізувати стабільність виконання завдань, визначати слабкі та сильні пари, прогнозувати рівень успішності та формувати індивідуальні рекомендації.

Гнучкість структури pairsStats також дозволяє легко розширювати модель новими показниками — наприклад, метриками ентропії, часовими індикаторами або поведінковими характеристиками, — що робить зберігання аналітичних даних масштабованим і придатним для подальшого розвитку алгоритмів адаптації.



```
router.put('/settings', protect, async (req, res) => {
  try {
    const { animations, animationSpeed, showHints, autoSave } = req.body;

    console.log('Updating settings:', { animations, animationSpeed, showHints, autoSave });


    const user = await User.findById(req.user._id);
    if (!user) {
      return res.status(404).json({ message: 'Користувача не знайдено' });
    }

    if (!user.settings) {
      user.settings = {
        animations: true,
        animationSpeed: 'normal',
        showHints: true,
        autoSave: true,
      };
    }

    if (animations !== undefined) user.settings.animations = animations;
    if (animationSpeed !== undefined) user.settings.animationSpeed = animationSpeed;
    if (showHints !== undefined) user.settings.showHints = showHints;
    if (autoSave !== undefined) user.settings.autoSave = autoSave;

    user.markModified('settings');

    await user.save();

    console.log('Settings saved:', user.settings);

    const userObj = user.toJSON();
    res.json({
      settings: userObj.settings,
    });
  } catch (error) {
    console.error('Error updating settings:', error);
    res.status(500).json({ message: 'Помилка сервера', error: error.message });
  }
});
```

Рисунок 4.33 – pairsStats у моделі User.js

Узгоджена робота цих модулів створює повний адаптивний цикл:

1. Користувач проходить ігровий рівень.
2. Фронтенд обчислює статистику (microAnalysis, SM-2, Elo).
3. Дані надсилаються до бекенду.
4. Сервер оновлює модель користувача.
5. Наступна гра генерується з урахуванням усіх попередніх результатів.

Таким чином, Memory+ перетворюється з простого набору мініігор на повноцінний когнітивний тренажер, який адаптується під поведінку користувача та сприяє довготривалому закріпленню знань.

4.6. Система налаштувань користувача

Система налаштувань дозволяє адаптувати роботу застосунку Memory+ під індивідуальні потреби користувача. Налаштування впливають не лише на інтерфейс, але й на поведінкову логіку всієї адаптивної системи, оскільки зміни користувача зберігаються у моделі User.js, передаються між фронтендом і бекендом та впливають на генерацію ігрових рівнів.

Усі параметри зберігаються у двох компонентах застосунку:

- settings.js - локальний модуль з переліком можливих налаштувань;
- settings.jsx - інтерфейс користувача для зміни налаштувань, а також система автоматичного збереження.

Після зміни будь-якого параметра оновлена конфігурація передається через API-метод:

PUT /api/user/settings

та записується у полі settings моделі користувача. Це забезпечує постійну відповідність між інтерфейсом, роботою гри та індивідуальними уподобаннями.

Основні параметри системи налаштувань

1. Візуальні налаштування

- theme – вибір світлої/темної теми;
- cardStyle – варіанти зображень карток;
- animations – можливість увімкнути або вимкнути анімації під час гри.

Ці параметри використовуються на рівні компонентів фронтенду та впливають на зручність взаємодії користувача з тренажером.

2. Параметри складності

- рівень складності (Easy/Normal/Hard);
- контроль кількості карткових пар;
- частка слабких пар у грі.

Для алгоритмічної частини це означає:

- коригування роботи Elo,
- збільшення або зменшення частоти появи слабких пар,

- зміну темпу інтервальних повторень.

3. Система підказок (hints)

- кількість доступних підказок;
- ліміт використання на день;
- параметри “hintsData” (usedToday, lastDate).

Підказки інтегровані у адаптивні механізми: часте використання підказок знижує рейтинги, рідке - збільшує.

4. Механізм autoSave

Компонент Settings.jsx реалізує авто-збереження:

- при зміні будь-якого параметра
- після затримки (debounce) 300–500 мс
- надсилає оновлення на бекенд
- відображає статус “Налаштування збережені”

Це забезпечує плавність роботи інтерфейсу та зменшує кількість запитів до сервера.

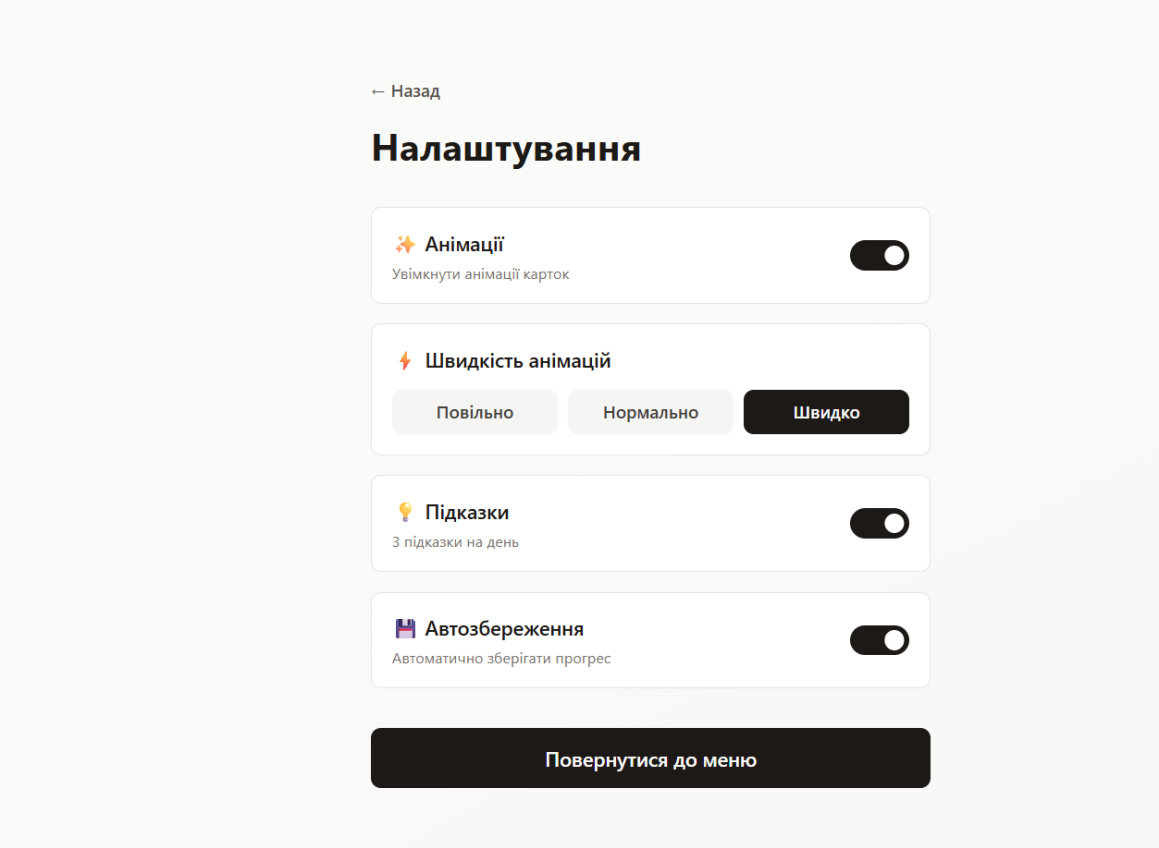


Рисунок 4.34 – Меню налаштувань

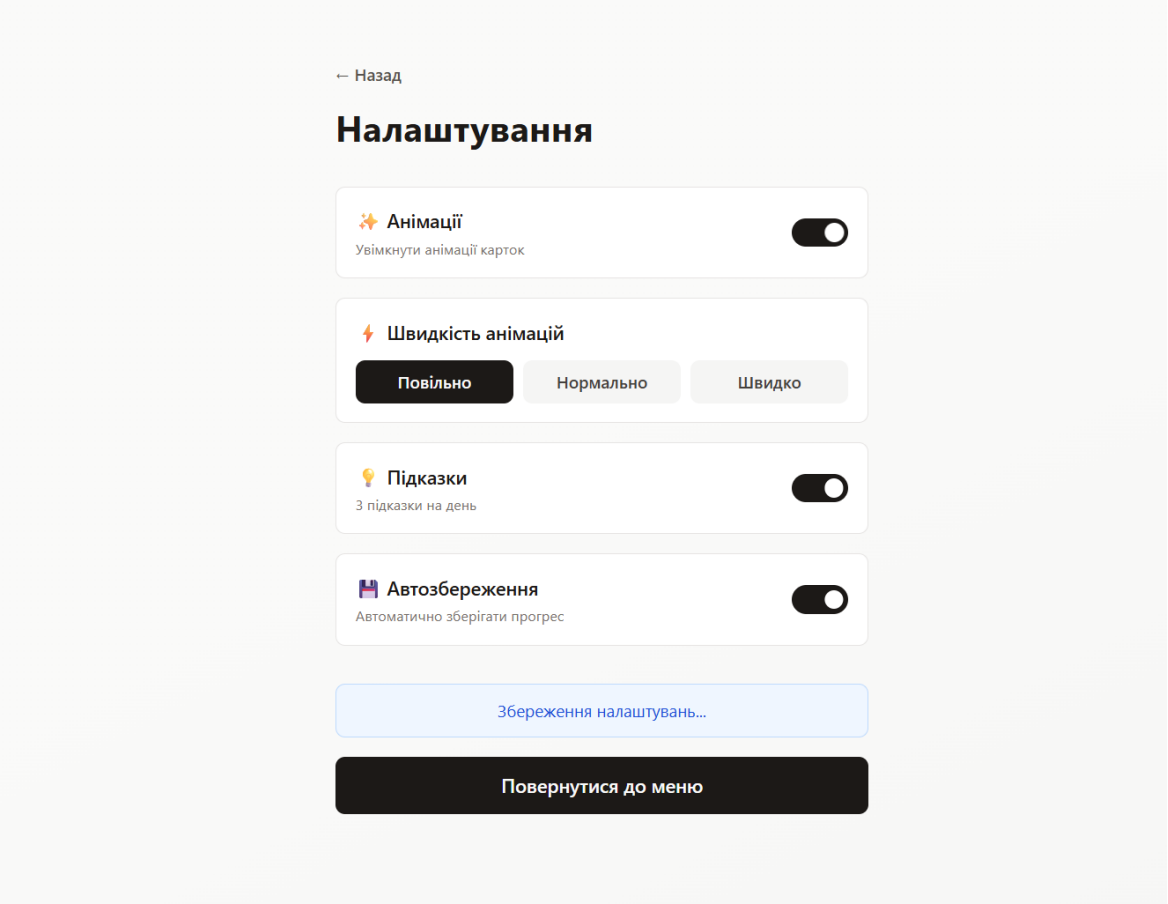


Рисунок 4.35 – Вікно збереження налаштувань

4.7. Тестування та налагодження

Процес тестування програмної системи Memory+ був спрямований на перевірку коректності роботи адаптивних алгоритмів, правильності функціонування API, стабільності фронтенду, а також цілісності взаємодії між клієнтом та сервером. Тестування проводилося багаторівнево: через консольні журнали розробника, відладку модулів, перевірку серверної логіки, тестування інтерфейсу та ручний контроль роботи адаптивних механізмів.

У проєкті застосовувалися внутрішні інструменти налагодження, вбудовані можливості браузера (DevTools), журнали серверної частини, а також зовнішні інструменти перевірки API, серед яких ключовим був Postman. Окрему увагу приділено сценаріям поведінкового тестування, що дозволяє оцінити роботу алгоритмів у реальних ігрових умовах.

Внутрішні інструменти налагодження

Під час розробки активно використовувалися консольні журнали:

- console.log у фронтенді для перевірки роботи microAnalysis, SM-2 та вибору карткових пар;
- console.error для відстеження неочікуваних помилок у логіці гри;
- журнали React DevTools для аналізу стану компонентів, пропсів та повторних рендерів;
- журнали Node.js у бекенді для відстеження логіки оновлення статистики, рейтингів та збереження даних у MongoDB.

Використання таких інструментів дозволило виявляти невідповідності у роботі алгоритмів, помилки обробки статистики та некоректні дані до моменту інтеграції фінальної логіки.

Обробка помилок

У серверній частині налаштовано централізований middleware для обробки помилок:

- перехоплення винятків у маршрутах,
- захист від некоректних або неповних запитів,
- обробка помилкових токенів авторизації,
- повернення стандартизованих JSON-повідомлень про помилки.

У фронтенді реалізовано локальні обробники у компонентах гри та налаштувань, що забезпечує стабільність інтерфейсу при некоректних даних або обриві з'єднання з сервером.

Перевірка API за допомогою Postman

Для тестування взаємодії між фронтендом і бекендом застосовувався Postman - інструмент, що дає змогу імітувати запити до серверних маршрутів. За його допомогою були протестовані:

- реєстрація та авторизація;
- отримання профілю;
- оновлення статистики / адаптивних параметрів;
- збереження налаштувань користувача;
- формування таблиці лідерів.

Це забезпечує повну перевірку правильності реалізації API та відповідності серверної логіки документації.

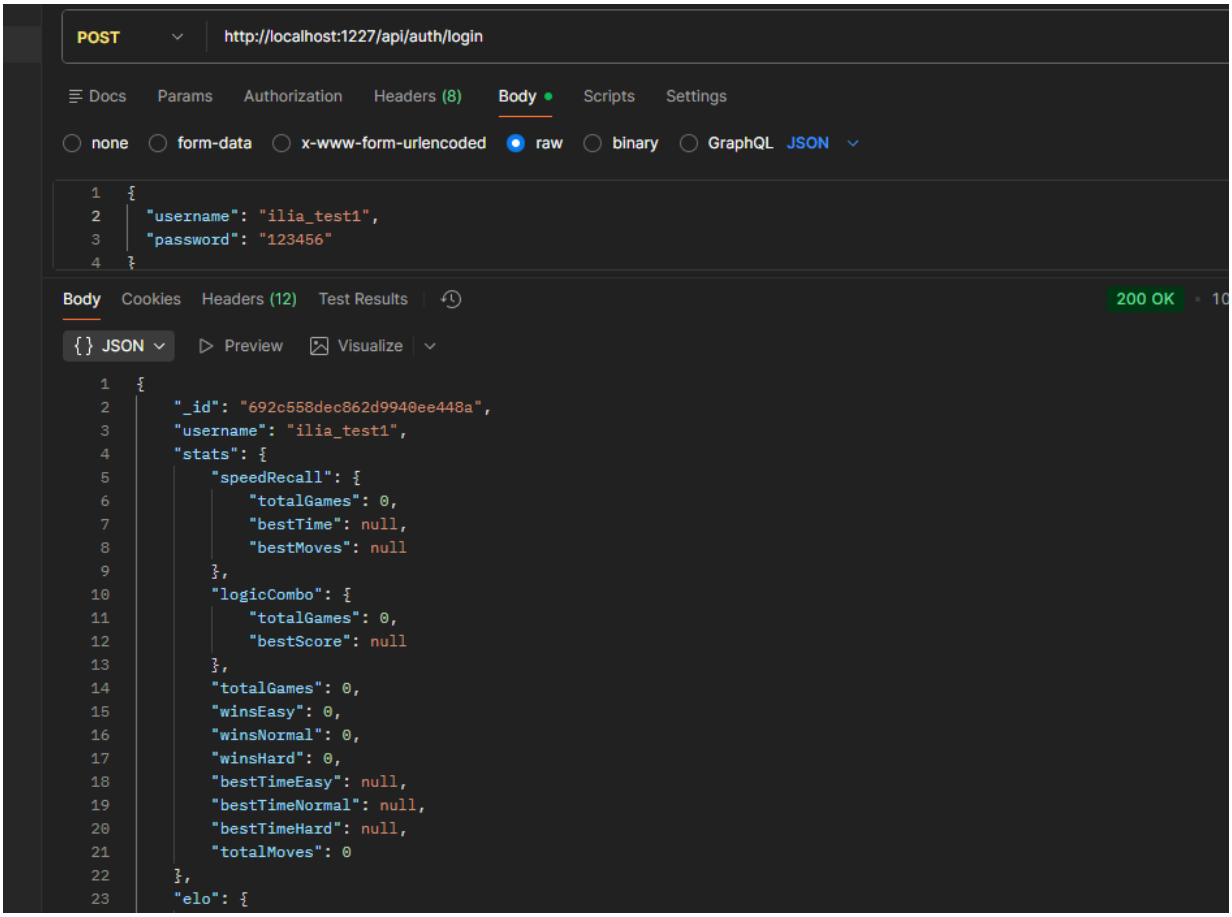


Рисунок 4.36 – Тестування API-кінцевої точки

Рисунок 4.37 – Тестування логіки адаптивних алгоритмів

Тестування логіки адаптивності

Під час внутрішнього тестування перевірялися:

- правильність роботи модуля microAnalysis (ймовірності, ентропія, прогноз);
- оновлення SM-2 після кожної спроби;
- зміна Elo згідно формули;
- формування weakPairs/strongPairs;
- вибір карткових пар залежно від рівня користувача.

Це дозволило переконатися, що система адаптації коректно реагує на ігрову поведінку та формує індивідуальну траєкторію навчання.

Поведінкове тестування інтерфейсу

Окрім тестування API та модулів логіки, проводилось ручне поведінкове тестування самої гри. Було перевірено:

- стабільність роботи на різних рівнях складності;
- відображення карток та анімацій;
- роботу підказок;
- інтеграцію нових наборів карт;
- коректність завершення гри та генерації статистики;
- вплив складності на вибір карток.

Поведінкові тести дозволили перевірити систему в реальних умовах та оцінити взаємодію між адаптивними алгоритмами і UI.

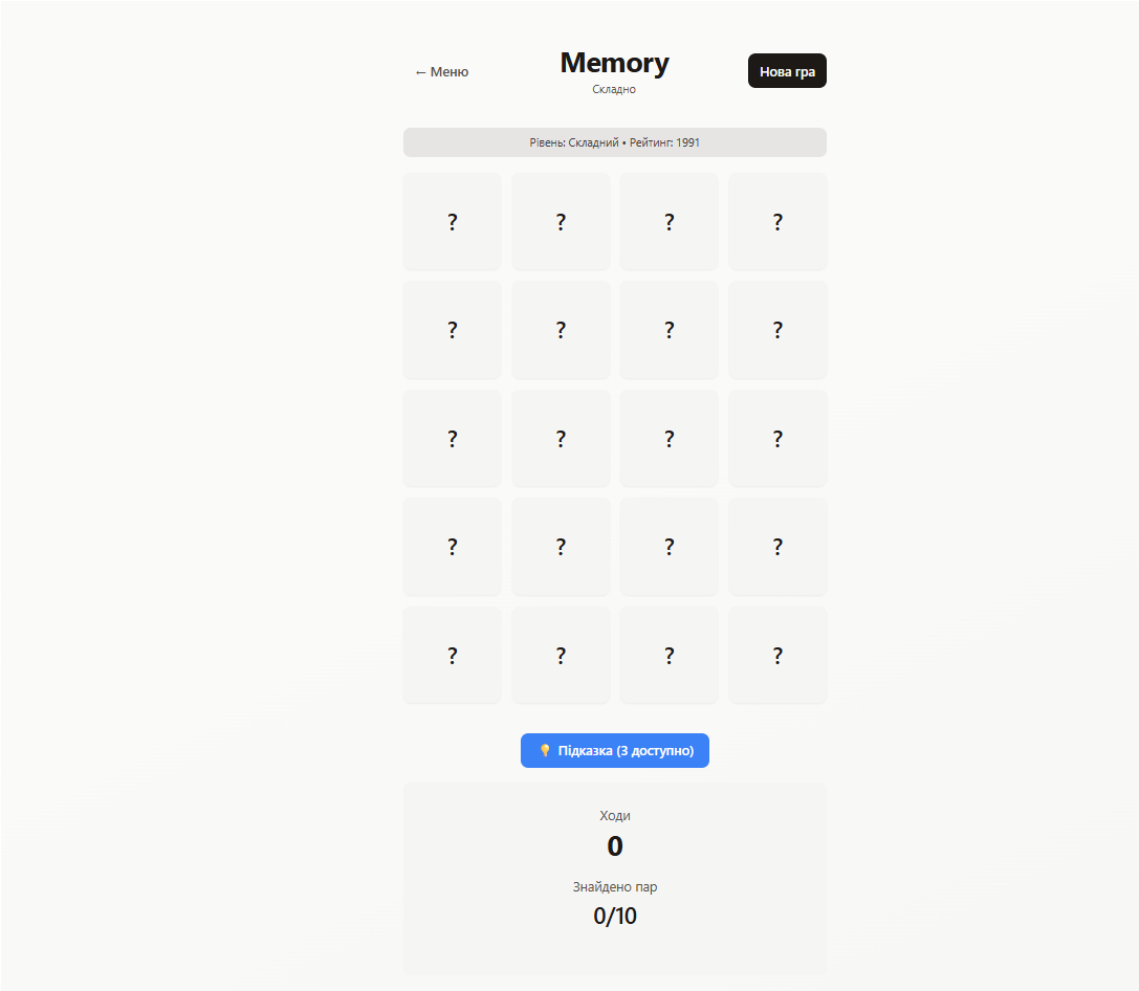


Рисунок 4.38 – Тестування рівнів складності у фронтенді

4.8. Контейнеризація та розгортання

Для забезпечення відтворюваності середовища, стабільності роботи та спрощення розгортання програмної системи Memory+ було застосовано технологію контейнеризації на основі Docker. У контейнерне оточення винесено три основні компоненти системи: клієнтську частину (frontend), серверну частину (backend) та базу даних MongoDB. Компоненти об’єднані у спільну мережеву інфраструктуру, що забезпечує їх взаємодію без необхідності використання фіксованих IP-адрес.

Контейнеризація використовується для ізоляції служб, мінімізації залежностей від операційної системи та можливості розгортання системи на будь-якому сервері, який підтримує Docker. Побудовані контейнери є автономними й можуть бути запущені однією командою за допомогою Docker Compose.

Dockerfile фронтенд-частини

Фронтенд зібраний на основі React/Vite і розміщується у контейнері разом із веб-сервером Nginx. Збірка виконується у два етапи:

1. Побудова статичних файлів у Node.js середовищі.
2. Розміщення результату у Nginx та запуск веб-сервера.

Такий підхід дозволяє розділити етапи збірки та виконання, зменшити розмір кінцевого образу та підвищити продуктивність у продакшні.

Dockerfile.frontend

```
1   FROM node:18-alpine as build
2
3   WORKDIR /app
4
5   # Copy package files
6   COPY package*.json ./
7
8   # Install dependencies
9   RUN npm ci
10
11  # Copy application files
12  COPY . .
13
14  # Build the application
15  RUN npm run build
16
17  # Output stage - just copy files
18  FROM alpine:latest
19  RUN apk --no-cache add cpio
20  WORKDIR /app
21  COPY --from=build /app/dist ./dist
22
23
```

Рисунок 4.39 – Dockerfile для фронтенда

Dockerfile бекенд-частини

Бекенд-підсистема працює на базі Node.js/Express. Dockerfile містить визначення робочої директорії, копіювання вихідного коду, встановлення залежностей та визначення порту, на якому працюватиме API. У середовищі контейнера сервер запускається в режимі, визначеному у файлі package.json.

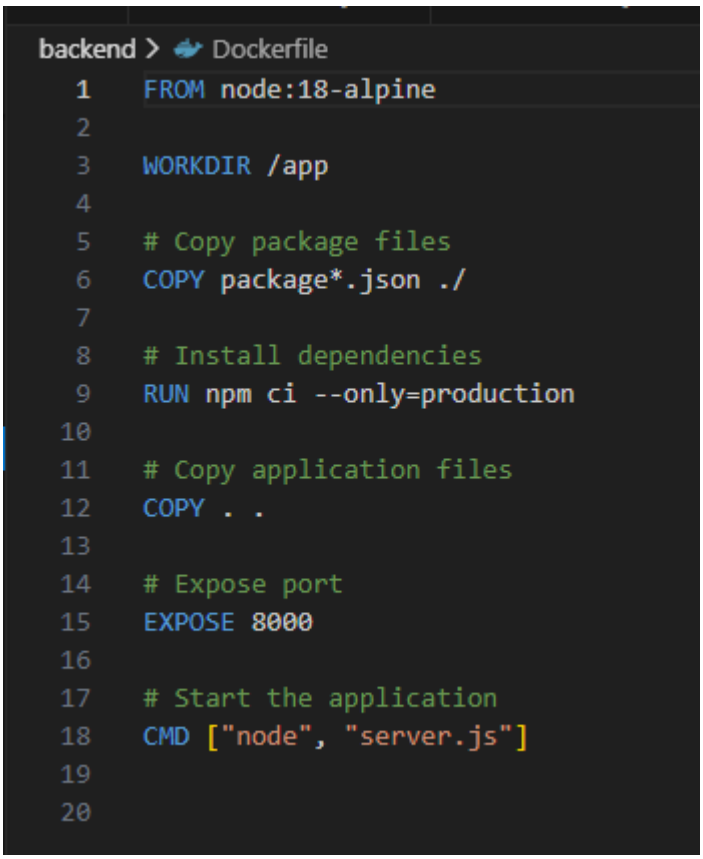


Рисунок 4.40 – Dockerfile для бекенда

Файл docker-compose.yml

Для запуску всієї інфраструктури використано docker-compose.yml, який описує три ключові сервіси:

- mongodb - контейнер із офіційним образом MongoDB;
- backend - контейнер із застосунком Node.js/Express;
- frontend - контейнер зі зібраним фронтендом на базі Nginx.

Усі сервіси підключені до спільної мережі memory-game-network, що забезпечує доступ до інших контейнерів за іменем сервісу. Так, бекенд звертається до бази даних через адресу:

mongodb://mongodb:27017/<database-name>

А фронтенд - до API через:

http://backend:1227

Обидва сервіси використовують змінні середовища, які передаються через файл .env, зокрема:

- PORT

- MONGO_URI
- JWT_SECRET
- параметри CORS
- адреси клієнтських доменів

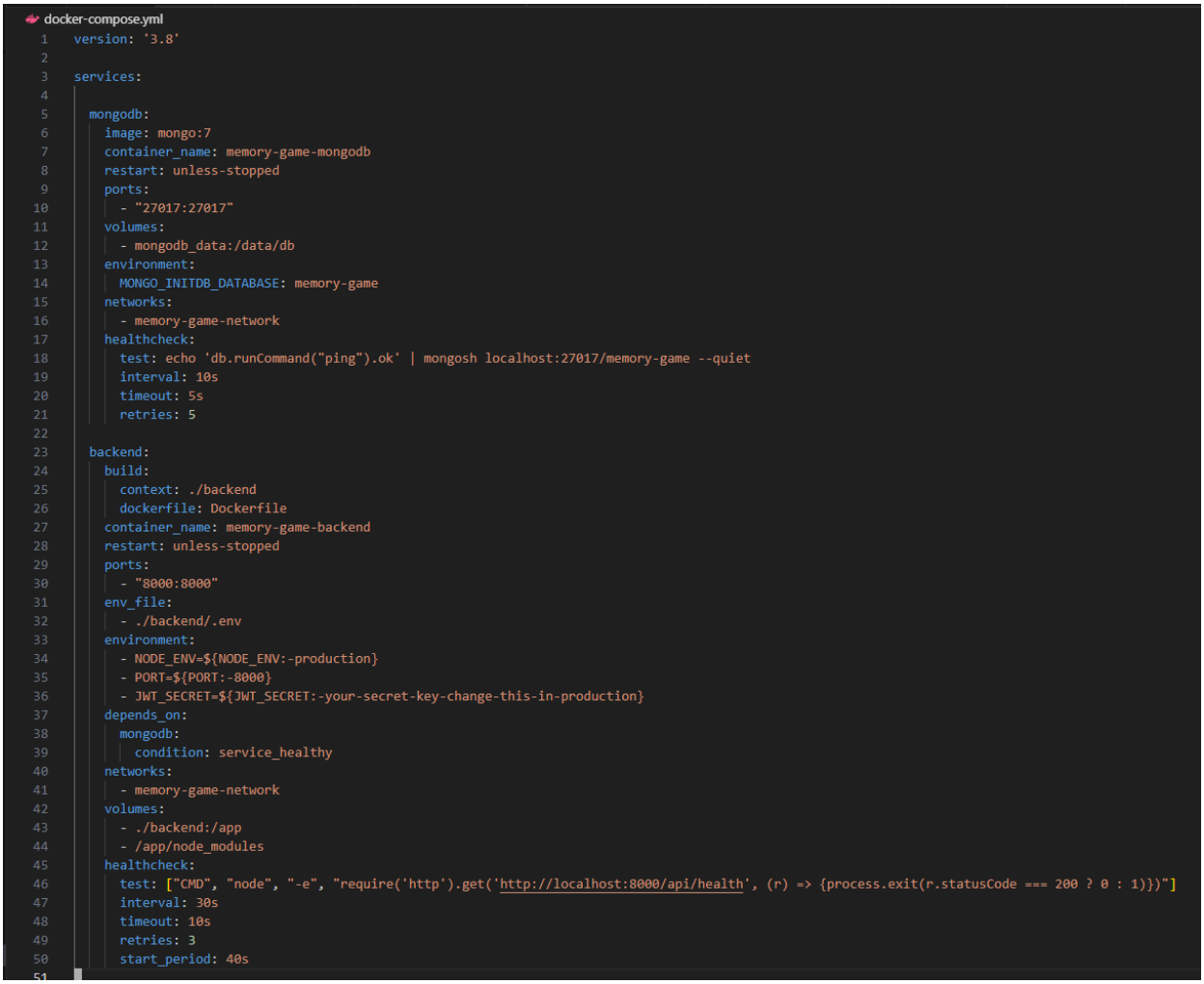


Рисунок 4.41 – docker-compose.yml проєкту Memory+

Розгортання через Docker Desktop

Локальне тестування та розгортання системи здійснюється за допомогою Docker Desktop. Інструмент дозволяє переглядати стан контейнерів, оглядати логи, перезапускати сервіси та контролювати використання ресурсів.

Після виконання команди:

docker-compose up –build

створюються такі контейнери:

- memory-game-backend

- memory-game-frontend
- memory-game-mongodb

Дані бази зберігаються у volume, що забезпечує їх стійкість до перезапусків.

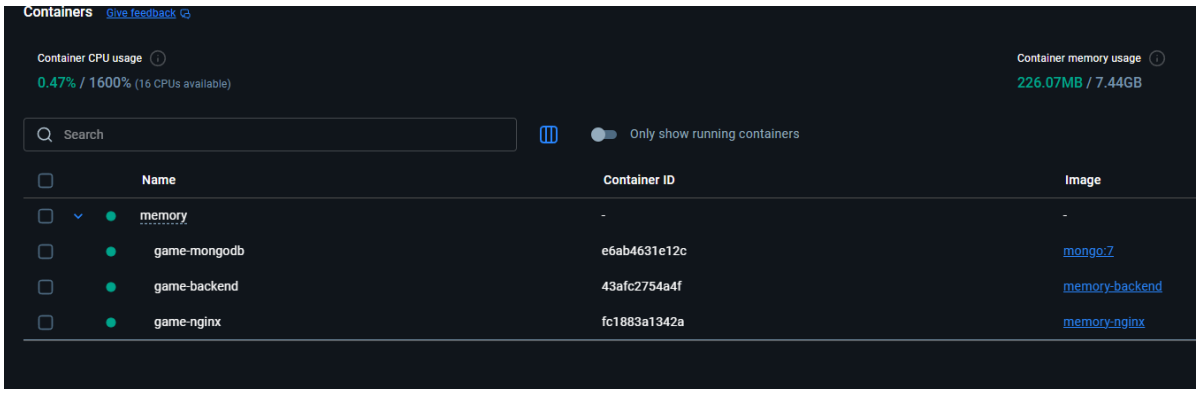


Рисунок 4.42 – Стан контейнерів Memory+ у Docker Desktop

Мережа контейнерів

Усі контейнери знаходяться у мережі типу bridge, яка створюється Docker Compose. Завдяки цьому контейнери взаємодіють між собою через DNS-імена:

- frontend → backend
- backend → mongodb

Це усуває потребу у використанні локальних портів усередині контейнерного середовища та робить архітектуру незалежною від хоста.

Nginx reverse proxy

У продакшн-режимі фронтенд обслуговується Nginx, який виконує такі функції:

- роздає статичні файли React/Vite;
- приймає всі HTTP-запити;
- маршрутизує запити /api/* у контейнер backend;
- може бути розширений для HTTPS-сертифікації.

Такий підхід забезпечує поділ логіки між серверами, захист бекенду та оптимізовану роботу з клієнтом.

Архітектура контейнерів системи Memory+

Нижче наведено загальну схему взаємодії контейнерів системи:

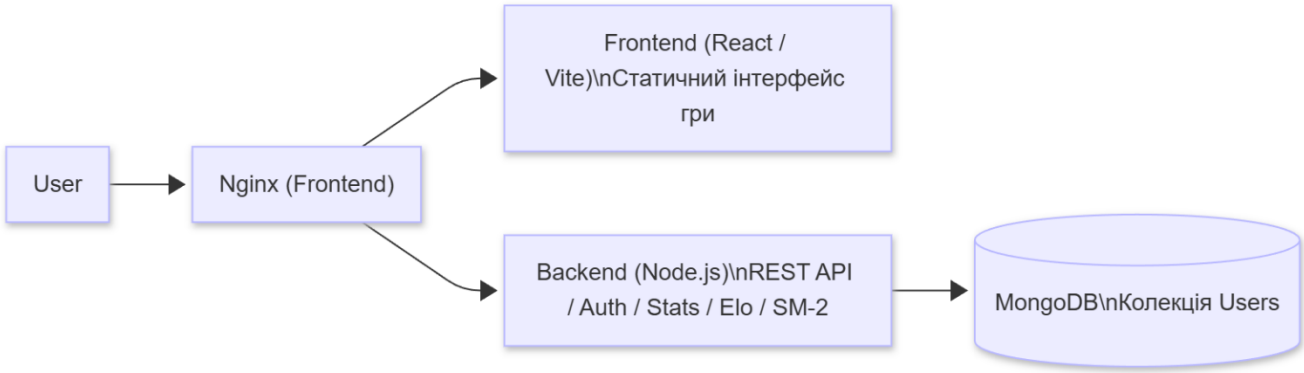


Рисунок 4.43 – Архітектура контейнерів Memory+

4.9. Висновки до розділу 4

У цьому розділі було здійснено комплексний опис програмної реалізації системи Memory+, яка включає фронтенд-, бекенд-компоненти, механізми адаптивності, модулі статистичного аналізу, систему збереження даних та контейнеризовану інфраструктуру для розгортання.

На стороні клієнта реалізовано інтерфейс для взаємодії з ігровими режимами, модулі обробки подій, динамічне формування рівнів складності, локальну логіку microAnalysis, алгоритм інтервального повторення SM-2 та обробку Elo-рейтингу. Фронтенд забезпечує формування ігрової статистики та передає її на сервер у стандартизованому форматі.

Серверна частина виконує функції автентифікації, приймання даних від клієнта, оновлення адаптивних параметрів, збереження статистики та формування таблиці лідерів. Модель користувача містить структури для збереження результатів ігрової діяльності, зокрема детальну інтерпретацію SM-2, параметри microAnalysis, ймовірнісні оцінки, прогнози та Elo-рейтинги для кожного режиму.

У процесі розроблення проведено тестування API-маршрутів, модулів логіки та поведінкових сценаріїв інтерфейсу. Було перевірено коректність обробки даних, стабільність взаємодії між клієнтом і сервером та відповідність адаптивних алгоритмів очікуваним результатам.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1 Загальна характеристика стартап-проєкту

Стартап-проєкт Memory+ є інноваційною освітньою платформою, спрямованою на розвиток та підтримку когнітивних здібностей користувачів за допомогою інтерактивних тренувальних модулів, математичного моделювання та персоналізованих алгоритмів адаптації. Основою проєкту є поєднання елементів гейміфікації, сучасних методів навчальної аналітики та теорії індивідуальних освітніх траєкторій, що дозволяє створити унікальний інструмент для тренування пам’яті, уваги, швидкості мислення та загальної інформаційної витривалості.

Потреба у створенні такого рішення обумовлена посиленням інформаційного навантаження на школярів та студентів, а також зростанням ролі самостійного навчання у сучасній освітній системі. Сучасні навчальні процеси часто не враховують індивідуальні когнітивні відмінності між учасниками, що призводить до зниження мотивації, неефективного запам’ятовування та хронічної перевтоми. Memory+ має на меті компенсувати недоліки традиційного підходу шляхом створення персоналізованого тренувального середовища.

У межах проєкту розроблено три основні ігрові режими - MatchCard, SpeedRecall та LogicCombo, кожен із яких тренує окремий компонент когнітивної діяльності: короткочасну пам’ять, швидкість відтворення інформації, логічне мислення та гнучкість уваги. На відміну від статичних навчальних інструментів, кожна гра у Memory+ працює на основі інтегрованої системи адаптивних алгоритмів. Зокрема, для оцінювання результатів користувача застосовано:

- рейтингову модель Elo, яка визначає індивідуальний рівень навантаження;
- ентропійний аналіз, що дозволяє оцінити непередбачуваність дій та когнітивну стабільність;
- експоненціальне згладжування, що відстежує короткострокову динаміку прогресу;

- алгоритм інтервального повторення SM-2, який формує оптимальні інтервали повторення для покращення довготривалої пам’яті.

Поєднання цих методів забезпечує комплексну підтримку користувача у формуванні навичок структурованого запам’ятовування. Таким чином, Memory+ виступає платформою, що поєднує навчальну функціональність із науково обґрунтованими підходами до формування індивідуальної освітньої траєкторії.

Окремою характеристикою стартапу є його технологічна архітектура. Система використовує сучасний стек веброзробки: React/Vite для клієнтської частини та Node.js з REST API для серверної логіки. База даних MongoDB Atlas забезпечує зберігання індивідуальних статистичних профілів, дозволяючи накопичувати та аналізувати великі масиви даних для подальшої адаптації ігор. Такий підхід гарантує масштабованість продукту, стабільність роботи та можливість подальшого розширення функціоналу.

Стартап Memory+ позиціонується передусім як продукт для шкільної та студентської аудиторії, що прагне підвищити власну продуктивність у засвоєнні навчального матеріалу, а також для користувачів, які хочуть підтримувати тренування пам’яті у повсякденному житті. Академічність застосованих математичних моделей у поєднанні з ігровими механіками роблять Memory+ не лише цікавим, але й науково обґрунтованим інструментом когнітивного розвитку.

Таким чином, стартап-проєкт Memory+ є комплексним рішенням, що поєднує теоретичні дослідження когнітивної психології, сучасні алгоритми адаптації та високотехнологічну вебреалізацію, формуючи інноваційний продукт у сфері EdTech, здатний вирішувати реальні освітні та пізнавальні потреби користувачів.

5.2 Аналіз ринку та конкурентного середовища

Ринок освітніх технологій (EdTech) у світі демонструє динамічне та стабільне зростання, що зумовлено модернізацією систем освіти, переходом до дистанційного та змішаного навчання, підвищенням вимог до самостійної роботи студентів та учнів, а також стрімким розвитком цифрової інфраструктури. За останні роки значно зріс попит на інструменти, які дозволяють підвищувати ефективність навчання та

формувати когнітивні навички. До таких навичок належать пам’ять, концентрація, швидкість опрацювання інформації, розподіл уваги та здатність до прийняття рішень.

Однією з ключових тенденцій у сфері EdTech є перехід від формату «універсальних» навчальних матеріалів до персоналізованих платформ, які підлаштовуються під користувача. Причиною цього є те, що сучасна молодь має різні когнітивні звички, рівні підготовки та темпи засвоєння. Масове застосування цифрових пристроїв, соціальних мереж та швидких форматів подання інформації впливає на спосіб навчання та потребує індивідуальних підходів. Водночас традиційні методики освіти не враховують цих відмінностей, що робить персоналізовані цифрові інструменти особливо актуальними.

Аналіз ринку показує, що сегмент тренажерів для розвитку пам’яті активно зростає, проте його більшість представлена мобільними або вебзастосунками зі статичними або частково персоналізованими механіками. Рішення, які пропонують глибоку адаптацію на основі математичних моделей, зустрічаються рідко, що формує ринкову нішу для Memory+.

Серед ключових гравців ринку варто виокремити такі продукти:

Lumosity. Один із найпопулярніших тренажерів для мозку, який пропонує великий набір вправ. Недоліком є те, що система використовує лише базові механізми адаптації. Вправи не аналізують детально поведінку користувача, а зміна складності залежить від загального прогресу, а не конкретних когнітивних параметрів.

Elevate. Містить вправи для розвитку пам’яті, логіки та читання. Алгоритми персоналізації описані поверхнево, а сам продукт орієнтується на широке коло користувачів без фокусування на студентській аудиторії.

Duolingo. Хоча це мовна платформа, її важливо згадати у контексті гейміфікації та адаптивності. Вона пропонує персональні траєкторії, проте сфера застосування вузька, а адаптація спрямована лише на корекцію рівня складності лексичних завдань.

Cognifit. Професійний інструмент для діагностики когнітивних здібностей, орієнтований на психологів і медичні установи. Його складність і специфічність роблять продукт малодоступним для шкільної аудиторії.

Незважаючи на наявність конкурентів, аналіз показує, що сучасні сервіси здебільшого не інтегрують такі складні й водночас ефективні методи адаптації, як модель Elo, ентропійний аналіз і алгоритм інтервального повторення SM-2. Більшість платформ не формують навчальну траєкторію на основі поведінкових даних користувача, а будують її за фіксованою схемою, що знижує ефективність тренувань.

Крім того, значна частина EdTech-платформ не орієнтована саме на школярів та студентів, тобто на аудиторію, яка найбільше потребує регулярного тренування пам'яті та концентрації у зв'язку з виконанням великої кількості навчальних завдань. Memory+ заповнює цю прогалину, пропонуючи зручний формат вебзастосунку, інтуїтивний інтерфейс, ігрову форму та глибоку адаптивність.

Додатковою конкурентною перевагою є можливість формування розширеної статистики тренувань та інтерпретації когнітивних показників. Це робить Memory+ не лише тренажером, а й інструментом аналізу індивідуальних навчальних стратегій. Система може використовуватися як автономно окремими користувачами, так і як допоміжний інструмент у школах, університетах або на освітніх курсах.

У підсумку, ринок EdTech створює сприятливі умови для запуску стартапу Memory+, оскільки він поєднує науково обґрунтовані методи адаптивності, широкі можливості персоналізації та орієнтацію на специфічні потреби молодіжної аудиторії. Наявні конкуренти формують лише широку галузеву конкуренцію, але не займають тієї конкретної ніші, яку покриває Memory+, що підтверджує ринкову доцільність та інноваційність проєкту.

5.3 Бізнес-модель та модель монетизації стартапу Memory+

Ефективна бізнес-модель є ключовою умовою життєздатності будь-якого стартапу, особливо у сфері освітніх технологій, де конкуренція висока, а користувачі очікують доступності, зручності та прозорості. Стартап Memory+ орієнтований на стійку довгострокову монетизацію, тому у його рамках обрано модель підписки (Subscription), яка є найбільш адаптивною для цифрових освітніх сервісів і забезпечує регулярний повторюваний дохід.

Модель підписки у Memory+ передбачає доступ користувачів до всього або частини функціоналу платформи залежно від обраного тарифного плану. Основна ідея

такої моделі полягає в тому, що користувач отримує можливість постійного вдосконалення своїх когнітивних навичок за умови регулярного використання сервісу, а платформа - стабільний потік доходів, що дозволяє розвивати продукт.

У межах Memory+ виділяються такі рівні доступу:

1. Базовий (безкоштовний) план

Цей тариф забезпечує мінімально необхідний функціонал для залучення нових користувачів та створення первинної цінності. У нього входять:

- обмежений доступ до ігрових режимів;
- базові статистичні звіти;
- щоденні тренування з мінімальним лімітом;
- можливість ознайомитися з адаптивністю системи без повного доступу до алгоритмів.

Мета цього тарифу - продемонструвати користувачам цінність продукту та мотивувати їх перейти до розширених можливостей.

2. Преміум-підписка (місячна або річна)

Це основний джерело монетизації. Преміум-користувачі отримують:

- повний доступ до усіх ігрових сценаріїв;
- розширену систему статистики, включаючи ентропію, показники прогнозування, історію Elo-рейтингів;
- індивідуальні рекомендації, побудовані на основі SM-2 та короткострокових моделей продуктивності;
- відсутність будь-якої реклами;
- доступ до нових модулів без додаткових оплат.

Преміум-підписка дозволяє користувачам отримати максимальну користь від Memory+, оскільки ключові алгоритми адаптивності найефективніше працюють саме при повному доступі до ігрових модулів.

3. Освітні тарифні пакети

Цей рівень призначений для:

- шкіл;

- університетів;
- навчальних центрів;
- групових курсів.

Освітній тариф передбачає доступ для групи користувачів, а також:

- адміністрування класів або потоку студентів;
- отримання зведених статистичних звітів;
- можливість інтеграції з внутрішніми освітніми платформами;
- нижчу вартість за одного користувача при великій кількості ліцензій.

Цей формат розширює ринок Memory+ та відкриває можливості для співпраці з освітніми установами на постійній основі.

Переваги моделі підписки для Memory+

1. Регулярний та прогнозований дохід

Підписки забезпечують стабільний фінансовий потік, що дозволяє планувати довгостроковий розвиток платформи-додавання нових ігор, покращення алгоритмів адаптивності, розширення аналітики.

2. Низький поріг входу

Користувач може спробувати базову версію безкоштовно, не ризикуючи фінансово. Це стимулює природне органічне зростання аудиторії.

3. Підтримка високої залученості користувачів

Оскільки платформа пропонує регулярні тренування та рекомендації, користувачам вигідно залишатися підписаними для збереження прогресу та стимуляції мозкової активності.

4. Масштабованість

Чим більше користувачів - тим вигідніша економіка продукту. Один бекенд може обслуговувати значну кількість користувачів без істотного збільшення витрат.

5. Можливість розширеної монетизації у майбутньому

Модель підписки дозволяє безболісно:

- додавати нові модулі до преміум-доступу;
- сегментувати користувачів за рівнями;

- запроваджувати сімейні або групові підписки;
- розширювати навчальні програми.

Особливості монетизації на ринку шкільної та студентської аудиторії

Оскільки основна цільова аудиторія - школярі та студенти, при формуванні тарифів важливо враховувати:

- доступність ціни (батьки/студенти мають бути готові платити);
- психологічне сприйняття підписок;
- важливість мотиваційних механік для утримання користувачів;
- гнучкість моделей (місячна, семестрова, річна підписка).

Підписна модель дозволяє адаптуватися до сезонності, характерної для освіти (початок навчального року, підготовка до екзаменів, сесій), створюючи додаткові маркетингові переваги.

Підсумок

Модель підписки для Memory+ є найбільш доцільним варіантом монетизації, оскільки:

- відповідає характеру продукту;
- легко масштабується;
- забезпечує довгостроковий фінансовий потік;
- створює умови для стабільної взаємодії з користувачами;
- дозволяє формувати різні рівні доступу;
- підтримує можливість подальших розширень.

5.4 Цільова аудиторія та ціннісна пропозиція продукту

Цільова аудиторія стартап-проєкту Memory+ визначається на перетині освітніх потреб, психологічних особливостей користувачів та їхньої готовності використовувати цифрові інструменти для саморозвитку. Оскільки основною метою системи є тренування пам’яті, концентрації та когнітивної продуктивності, важливою передумовою формування аудиторії є наявність мотиваційної та навчальної потреби у регулярних тренуваннях. У рамках цього проєкту головним сегментом користувачів

визначено школярів і студентів, які становлять найчисельнішу та найбільш активно зростаючу групу на ринку EdTech.

Школярі середніх та старших класів активно використовують цифрові технології для навчання та відпочинку, мають високі вимоги до інтерактивності та швидкості взаємодії з продуктом. Цей сегмент характеризується нестабільністю уваги, швидкою втомлюваністю та необхідністю постійного повторення навчального матеріалу. Memory+ задовольняє їхні потреби завдяки гейміфікованому підходу, адаптивним алгоритмам, щоденним тренуванням та зручній аналітиці, яка дозволяє бачити прогрес у реальному часі.

Студенти, у свою чергу, стикаються з необхідністю опрацьовувати великі обсяги інформації, що стає дедалі складніше у сучасних умовах, коли загальне інформаційне середовище перенасичене. Вони прагнуть підвищити ефективність навчання, покращити робочу пам’ять, швидкість реакції, здатність до зосередження та переробки даних. Memory+ пропонує їм інструмент, який органічно поєднується з навчальною діяльністю та дозволяє тренувати когнітивні здібності в будь-який зручний час.

Важливим елементом визначення цільової аудиторії є аналіз поведінкових факторів. Школярі та студенти надають перевагу коротким та динамічним форматам взаємодії з інформацією, мають високу чутливість до ігрових механік та прагнуть бачити відчутний прогрес. Система Memory+ враховує ці особливості, пропонуючи мотивуючі візуальні елементи, систему досягнень, рівнів, щоденних викликів, а також рекомендації, що ґрунтуються на попередній активності користувача.

Ще одним фактором формування цільової аудиторії є психологічна привабливість персоналізованих рішень. Користувачі сучасного покоління очікують, що цифрові інструменти підлаштовуватимуться під них, а не навпаки. Одним із ключових елементів ціннісної пропозиції Memory+ є саме індивідуальна адаптація - система автоматично коригує складність ігрових завдань, інтервали повторення та тип вправ залежно від результатів конкретного користувача. Це робить продукт відчутно ефективнішим, ніж звичайні тренажери, які пропонують усім однаковий рівень складності.

Ціннісна пропозиція Memory+ полягає у можливості створення персоналізованої траєкторії когнітивного розвитку. Платформа не просто пропонує тренування, а аналізує поведінку користувача, відстежує динаміку прогресу, визначає слабкі та сильні сторони, формує рекомендації та адаптує систему навчання. Такий підхід забезпечує:

–зростання ефективності тренувань, оскільки навантаження відповідає реальним здібностям користувача;

–підвищення зацікавленості, оскільки ігрові механіки роблять процес тренувань приємним і мотиваційним;

– покращення навчальних результатів, оскільки регулярні тренування пам’яті та уваги є основою для успішного засвоєння будь-яких предметів.

Окрім індивідуальних користувачів, важливим сегментом є освітні установи. Memory+ може застосовуватися як допоміжний інструмент у школах, коледжах та університетах для підтримки психологічної стійкості студентів, покращення продуктивності та формування навичок концентрації. Платформа дає можливість викладачам отримувати узагальнену статистику, використовувати ігри як елемент підготовки до уроків або літературно-логічних занять.

Таким чином, Memory+ має широку цільову аудиторію, але водночас чітко орієнтується на сегмент молоді, яка є найбільш відкритою до використання сучасних EdTech-рішень. Ціннісна пропозиція продукту ґрунтується на поєднанні когнітивної ефективності, наукової обґрунтованості та зручності використання, що робить Memory+ конкурентоспроможним та актуальним інструментом для освітнього середовища.

5.5 Маркетингова стратегія та план впровадження стартапу Memory+

Маркетингова стратегія стартапу Memory+ визначає комплекс заходів, спрямованих на залучення, утримання та перетворення користувачів у постійних підписників. Оскільки проєкт орієнтований на школярів та студентів - найбільш активну цифрову аудиторію - маркетингова діяльність повинна враховувати особливості поведінки цієї групи, їхні мотиваційні фактори та канали інформаційної

взаємодії. Стратегія просування має базуватися на поєднанні цифрових каналів, гейміфікаційних підходів, рекомендаційних систем та партнерських програм.

Цифровий маркетинг та позиціювання

Основним середовищем для просування Memory+ є соціальні мережі, де молодь проводить значну частину часу. Продукт має бути представлений у найпопулярніших каналах комунікації - TikTok, Instagram, YouTube та Telegram. Формат подання контенту має бути максимально зрозумілим, динамічним та візуально привабливим. Короткі відеоролики з демонстрацією прогресу у грі, порівнянням результатів, порадами щодо тренування пам'яті та гумористичним контентом здатні швидко поширюватися серед підлітків та студентів.

Рекламні кампанії у TikTok та Instagram повинні акцентувати увагу на простоті використання платформи, її адаптивності, результативності та можливості побачити свій прогрес у реальному часі. Особливий наголос робиться на тому, що Memory+ допомагає покращити навчальні результати, готуватися до контрольних робіт, екзаменів та сесії.

SEO та контент-маркетинг

Окремим напрямом є створення освітнього контенту, орієнтованого на запити користувачів щодо технік запам'ятовування, покращення концентрації, подолання прокрастинації, підготовки до іспитів та розвитку когнітивних функцій. Публікація тематичних статей на блозі платформи та оптимізація їх під пошукові системи (SEO) забезпечить додатковий органічний трафік. Це також підвищить авторитетність платформи та довіру з боку користувачів.

Гейміфікація як елемент маркетингової стратегії

Гейміфікаційні механіки не лише стимулюють користувача до регулярних тренувань, а й виконують маркетингову функцію. Система досягнень, щоденних завдань, streak-серій, нагород, рейтингових таблиць та персональних рекомендацій створює ефект залучення та підсилює мотивацію. Сучасні молоді люди реагують на регулярні "виклики", тому прив'язка до щоденного тренування сприяє збільшенню часу, проведеного у системі, а також росту конверсії у преміум-підписку.

Соціальні механіки - можливість порівнювати результати з друзями, брати участь у внутрішніх змаганнях, досягати нових рівнів - створюють природну вірусність продукту. Користувачі охоче діляться своїми здобутками у соцмережах, що забезпечує органічне поширення серед широкої аудиторії.

Партнерські програми та співпраця з освітніми установами

Одним з ключових напрямів розвитку Memory+ є співпраця зі школами, університетами, репетиторами та освітніми платформами. Освітні заклади можуть використовувати Memory+ як допоміжний інструмент для розвитку пам'яті та уваги учнів, підтримки психологічної стійкості та формування навичок самостійного навчання.

У рамках партнерських програм можливе:

- надання групових підписок;
- формування зведених статистичних звітів для викладачів;
- інтеграція тренажерів у навчальні програми;
- проведення турнірів, конкурсу успішності, навчальних марафонів.

Цей напрям сприяє формуванню стабільної клієнтської бази та розширенню присутності Memory+ на освітньому ринку.

План упровадження Memory+

План упровадження передбачає кілька етапів:

1. Запуск MVP-версії.

Основна мета - отримати зворотний зв'язок від перших користувачів, виявити технічні недоліки та перевірити працездатність ключових алгоритмів.

2. Бета-версія.

На цьому етапі додаються нові ігрові режими, поглиблюється аналітика, оптимізується інтерфейс, удосконалюється адаптивність.

3. Основний реліз.

Включає запуск підписки, масштабну маркетингову кампанію, залучення партнерів та інтеграцію у навчальні процеси.

4. Масштабування.

Додавання нових когнітивних тренажерів, розширення функціоналу статистики, розробка мобільної версії, вихід на міжнародні ринки.

Запуск Memory+ передбачає комплексне поєднання технічних, маркетингових та освітніх інструментів, які разом формують стратегічно стійкий шлях розвитку продукту.

Підсумок

Маркетингова стратегія Memory+ базується на сучасних цифрових практиках, орієнтованих на молодіжну аудиторію, та поєднує гейміфікацію, SEO, соціальні мережі, партнерства й адаптивність продукту. Такий підхід дозволяє формувати стабільний інтерес до платформи, ефективно утримувати користувачів та забезпечувати високий рівень конверсії у преміум-підписку.

5.6 Висновки до розділу 5

У цьому розділі було проведено комплексний аналіз та розроблено стартап-проєкт Memory+ як інноваційний продукт у сфері освітніх технологій. Здійснена характеристика проєкту показала, що Memory+ базується на поєднанні сучасних вебтехнологій, математичних моделей адаптивності та гейміфікаційних механік, що забезпечує високу ефективність тренування когнітивних навичок. Адаптивні алгоритми, зокрема рейтингові моделі, ентропійний аналіз, прогнозування та інтервальне повторення, формують фундамент індивідуальної освітньої траєкторії кожного користувача.

Проведений ринковий аналіз підтвердив актуальність і перспективність Memory+ у сегменті EdTech, де відчувається нестача платформ, здатних забезпечувати персоналізоване тренування пам’яті та уваги. Ідентифіковано, що наявні конкуренти не використовують багатошарову адаптивність, яка включає як поведінкові, так і статистичні компоненти, що дає Memory+ конкурентну перевагу.

Розроблена бізнес-модель підтверджує економічну обґрунтованість платформи. Модель підписки забезпечує стабільний повторюваний дохід і дозволяє ефективно масштабувати продукт, тоді як освітні тарифні плани відкривають можливість співпраці з навчальними закладами та створення корпоративних рішень. Чітко

окреслена цільова аудиторія - школярі та студенти - формує орієнтир для адаптації контенту, маркетингової стратегії та формування ціннісної пропозиції.

Запропонована маркетингова стратегія демонструє комплексний підхід до просування Memory+, який включає цифровий маркетинг, SEO, соціальні мережі, гейміфікацію та партнерські програми з освітніми установами. Такий підхід забезпечує достатню гнучкість для виходу на різні сегменти ринку і сприяє підвищенню впізнаваності бренду.

Дорожня карта впровадження Memory+ визначає послідовні етапи запуску - від MVP до масштабування та виходу на міжнародні ринки. Це створює чітку стратегію розвитку продукту та дозволяє прогнозувати його функціональне зростання.

Узагальнюючи результати проведеної роботи, можна стверджувати, що стартап Memory+ має високий інноваційний потенціал, відповідає сучасним тенденціям розвитку освітніх технологій, вирішує реальні потреби користувачів і демонструє значні перспективи подальшої комерціалізації та масштабування. Розроблені концепції та моделі можуть слугувати основою для подальшого розвитку платформи та її інтеграції в освітні процеси різного рівня.

ВИСНОВКИ

У магістерській роботі виконано комплексне дослідження, спрямоване на моделювання адаптивного рівня складності та ефективності запам’ятовування у навчальних іграх. На основі теоретичного аналізу моделей пам’яті, механізмів когнітивного навчання та сучасних математичних підходів до оцінювання складності було створено інтерактивну веб-платформу Memory+, що поєднує елементи гейміфікації та адаптивні алгоритми навчання.

Проведений огляд наукових джерел засвідчив, що використання адаптивних систем у комп’ютерних навчальних середовищах є одним із найефективніших підходів для формування та підтримання когнітивних навичок. На основі цього було сформовано вимоги до системи, обрано відповідні математичні моделі та визначено компоненти, які дозволяють інтегрувати адаптивність у процес тренування пам’яті.

У межах роботи було розроблено математичне забезпечення системи, до складу якого увійшли:

– модифікована система Elo для автоматичної зміни рівня складності;

– алгоритм експоненціального згладжування для прогнозування майбутньої успішності;

– ентропійна оцінка складності ігрового поля;

– алгоритм інтервального повторення SM-2 для формування довготривалих стратегій навчання.

Запропоновані моделі інтегровано у програмну реалізацію системи. На фронтенді реалізовано модулі microAnalysis, spacedRepetition та eloSystem, які забезпечують аналіз ігрових подій і формування індивідуальної траєкторії користувача. Бекендова частина приймає, обробляє та зберігає статистику, формує таблицю лідерів, виконує авторизацію та підтримує структуру даних, необхідну для адаптації. Дані користувачів зберігаються у базі MongoDB за допомогою моделі User, яка містить усі параметри, необхідні для адаптивного навчання.

Для забезпечення портативності та масштабованості розроблено контейнеризовану інфраструктуру Memory+, що включає окремі контейнери для

фронтенду, бекенду та бази даних. Розгортання відбувається за допомогою Docker та Docker Compose, а статичний контент обслуговується через Nginx.

Проведене тестування підтвердило коректність роботи алгоритмів адаптації, стабільність API, відповідність даних між фронтендом і бекендом, а також коректність збереження та обробки статистики. Поведінкові тести засвідчили, що система динамічно змінює складність гри залежно від успішності користувача, а також формує умови для повторення складних карткових пар.

У результаті роботи створено веб-платформу, яка поєднує механізми гейміфікації з математично обґрунтованими адаптивними методами навчання. Розроблена система Memory+ може бути використана як тренажер пам’яті, інструмент для когнітивної реабілітації або компонент ширших навчальних платформ. Запропоновані підходи можуть бути розширені та застосовані для інших жанрів навчальних ігор, що відкриває перспективи подальших досліджень у напрямку адаптивних освітніх технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Anderson J. R. Learning and Memory: An Integrated Approach. — Hoboken: Wiley, 2015. — 448 p.
2. Baddeley A. D., Hitch G. Working Memory and Cognition. — Oxford: Oxford University Press, 2020. — 312 p.
3. Karpicke J., Roediger H. Retrieval-Based Learning: Active Retrieval Promotes Meaningful Learning // Current Directions in Psychological Science. — 2012. — Vol. 21. — P. 157–163.
4. Bjork E., Bjork R. Desirable Difficulties in Theory and Practice // Journal of Applied Research in Memory and Cognition. — 2020.
5. Pavlik P. I., Anderson J. Practice and Forgetting: Designing Optimally Efficient Learning Schedules // Cognitive Science. — 2008. — P. 559–610.
6. Mayer R. Multimedia Learning. 3rd ed. — Cambridge: Cambridge University Press, 2021. — 786 p.
7. Burkov A. The Hundred-Page Machine Learning Book. — Andriy Burkov, 2019. — 160 p.
8. Montgomery D. C. Applied Statistics and Probability for Engineers. — Hoboken: Wiley, 2019. — 912 p.
9. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning. 2nd ed. — Springer, 2020. — 745 p.
10. Sutton R. S., Barto A. G. Reinforcement Learning: An Introduction. 2nd ed. — MIT Press, 2020. — 564 p.
11. Goodfellow I., Bengio Y., Courville A. Deep Learning. — MIT Press, 2016. — 800 p.
12. Silver D. UCL Reinforcement Learning Lectures [Електронний ресурс]. — 2020. — Режим доступу: https://www.deepmind.com/learning
13. Glickman M. Example of Adaptive Rating Systems Beyond Elo // Journal of Quantitative Analysis in Sports. — 2019.
14. Elo A. The Modern Elo System Revisited // ACM Transactions on Applied Analytics. — 2016.

15. Nielsen J., Budiu R. Mobile Usability. — New Riders, 2013.
16. Baxter K., Courage C., Caine K. Understanding Your Users. 3rd ed. — Morgan Kaufmann, 2015.
17. Tidwell J. Designing Interfaces: Patterns for Effective Interaction Design. 3rd ed. — O'Reilly, 2020.
18. Shneiderman B., Plaisant C., Cohen M. Designing the User Interface. 6th ed. — Pearson, 2022.
19. React Documentation [Електронний ресурс]. — Режим доступу: https://react.dev
20. Node.js Documentation [Електронний ресурс]. — Режим доступу: https://nodejs.org
21. Express.js Documentation [Електронний ресурс]. — Режим доступу: https://expressjs.com
22. Vite Documentation [Електронний ресурс]. — Режим доступу: https://vitejs.dev
23. MongoDB Documentation [Електронний ресурс]. — Режим доступу: https://www.mongodb.com/docs
24. Docker Documentation [Електронний ресурс]. — Режим доступу: https://docs.docker.com
25. Nginx Documentation [Електронний ресурс]. — Режим доступу: https://nginx.org
26. Ngrok Documentation [Електронний ресурс]. — Режим доступу: https://ngrok.com/docs
27. Postman Learning Center [Електронний ресурс]. — Режим доступу: https://learning.postman.com
28. Wickens C. D. Engineering Psychology and Human Performance. — Psychology Press, 2020. — 592 p.
29. OpenAI Research. Technical Reports on Adaptive Learning Models [Електронний ресурс]. — 2023. — Режим доступу: https://openai.com/research

ДОДАТОК А - КОД ФРОНТЕНДУ СИСТЕМИ MEMORY+

Основні компоненти:

GameBoard.jsx

```
import GameCard from "./GameCard";

function GameBoard({ cards, onCardClick, selectedCards }) {
  return (
    <div className="grid grid-cols-4 gap-3 mb-8">
      {cards.map((card) => (
        <GameCard
          key={card.id}
          card={card}
          isSelected={selectedCards.includes(card.id)}
          onClick={() => onCardClick(card.id)}
        />
      ))}
    </div>
  );
}

export default GameBoard;
```

GameCard.jsx

```
import { isAnimationsEnabled, getAnimationDuration } from "../utils/settings";

function GameCard({ card, isSelected, onClick }) {
  const animationsEnabled = isAnimationsEnabled();
  const animationDuration = getAnimationDuration();

  // Формуємо класи для анімації
  const transitionClass = animationsEnabled
    ? `transition-all`
    : '';
  const durationStyle = animationsEnabled
    ? { transitionDuration: `${animationDuration}ms` }
    : { transitionDuration: '0ms' };

  return (
    <button
      onClick={onClick}
```

```
        className={`aspect-square rounded-lg ${transitionClass} flex items-center justify-center text-2xl font-semibold cursor-pointer ${
          card.isMatched
            ? "bg-stone-200 text-stone-400 cursor-default"
            : card.isFlipped || isSelected
              ? "bg-stone-900 text-stone-50 shadow-md"
              : "bg-stone-100 text-stone-900 hover:bg-stone-200 shadow-sm"
        }`}
        style={durationStyle}
        disabled={card.isMatched}
      >
        {card.isFlipped || isSelected ? card.symbol : "?"}
      </button>
    );
  }

  export default GameCard;
```

GameHeader.jsx

```
  function GameHeader({ onNewGame, onBackToMenu, difficulty }) {
    const difficultyLabel = {
      easy: "Легко",
      normal: "Нормально",
      hard: "Складно",
    };

    return (
      <div className="flex items-center justify-between mb-8 gap-3">
        <button
          onClick={onBackToMenu}
          className="px-3 py-2 text-stone-600 hover:text-stone-900 transition-colors duration-200 text-sm font-medium"
          title="Back to menu"
        >
          ← Меню
        </button>

        <div className="text-center flex-1">
          <h1 className="text-3xl font-bold text-stone-900 tracking-tight">Memory</h1>
          {difficulty && <p className="text-xs text-stone-600 mt-1">{difficultyLabel[difficulty]}</p>}
        </div>
```

```
        <button
          onClick={onNewGame}
          className="px-3 py-2 bg-stone-900 text-stone-50 rounded-lg text-sm font-medium hover:bg-stone-800 transition-colors duration-200"
        >
          Нова гра
        </button>
      </div>
    );
  }

  export default GameHeader;

  GameStats.jsx
  function GameHeader({ onNewGame, onBackToMenu, difficulty }) {
    const difficultyLabel = {
      easy: "Легко",
      normal: "Нормально",
      hard: "Складно",
    };

    return (
      <div className="flex items-center justify-between mb-8 gap-3">
        <button
          onClick={onBackToMenu}
          className="px-3 py-2 text-stone-600 hover:text-stone-900 transition-colors duration-200 text-sm font-medium"
          title="Back to menu"
        >
          ← Меню
        </button>

        <div className="text-center flex-1">
          <h1 className="text-3xl font-bold text-stone-900 tracking-tight">Memory</h1>
          {difficulty && <p className="text-xs text-stone-600 mt-1">{difficultyLabel[difficulty]}</p>}
        </div>

        <button
          onClick={onNewGame}
          className="px-3 py-2 bg-stone-900 text-stone-50 rounded-lg text-sm font-medium hover:bg-stone-800 transition-colors duration-200"
```

```
      >
        Нова гра
      </button>
    </div>
  );
}

export default GameHeader;

Leaderboard.jsx
import { useState, useEffect } from "react";

const API_URL = '';

function LeaderBoard({ user, onBack }) {
  const [mode, setMode] = useState('classic');
  const [leaderboard, setLeaderboard] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState("");

  useEffect(() => {
    fetchLeaderboard();
  }, [mode]);

  const fetchLeaderboard = async () => {
    try {
      setLoading(true);
      setError("");
      const response = await fetch(`${API_URL}/api/leaderboard?mode=${mode}&limit=50`);

      if (!response.ok) {
        throw new Error('Не вдалося завантажити таблицю лідерів');
      }

      const data = await response.json();
      setLeaderboard(data.leaderboard || []);
    } catch (err) {
      console.error('Error fetching leaderboard:', err);
      setError(err.message || 'Помилка завантаження таблиці лідерів');
    } finally {
      setLoading(false);
```

```
  }
};

const formatTime = (seconds) => {
  if (seconds === null || seconds === 0) return "—";
  const mins = Math.floor(seconds / 60);
  const secs = seconds % 60;
  return mins > 0 ? `${mins}хв ${secs}с` : `${secs}с`;
};

const getModeLabel = (mode) => {
  switch (mode) {
    case 'classic':
      return '🎯 Match Card Game';
    case 'speedRecall':
      return '⚡ Speed Recall';
    case 'logicCombo':
      return '🧠 Memory + Logic Combo';
    default:
      return mode;
  }
};

const getRankIcon = (rank) => {
  if (rank === 1) return '🥇';
  if (rank === 2) return '🥈';
  if (rank === 3) return '🥉';
  return null;
};

const getCurrentUserRank = () => {
  if (!user) return null;
  const index = leaderboard.findIndex(entry => entry.username === user.username);
  return index >= 0 ? index + 1 : null;
};

const currentUserRank = getCurrentUserRank();
const currentUserEntry = leaderboard.find(entry => entry.username === user?.username);

return (
```

```
        <main className="min-h-screen bg-gradient-to-br from-stone-50 via-stone-50 to-stone-100 flex items-center justify-center p-4">
          <div className="w-full max-w-2xl">
            {/* Header */}
            <div className="mb-8">
              <button onClick={onBack} className="text-sm text-stone-600 hover:text-stone-900 transition mb-4 font-medium">
                ← Назад
              </button>
              <h1 className="font-sans text-3xl font-bold text-stone-900">Таблиця лідерів</h1>
            </div>

            {/* Mode Selector */}
            <div className="mb-6">
              <div className="flex gap-2 bg-white border border-stone-200 rounded-lg p-1">
                {['classic', 'speedRecall', 'logicCombo'].map((m) => (
                  <button
                    key={m}
                    onClick={() => setMode(m)}
                    className={`flex-1 py-2 px-3 rounded-lg text-sm font-medium transition ${
                      mode === m
                        ? 'bg-stone-900 text-stone-50'
                        : 'text-stone-700 hover:bg-stone-100'
                    }`}
                  >
                    {m === 'classic' ? '🎯' : m === 'speedRecall' ? '⚡' : '□'}
                  </button>
                ))}
              </div>
              <p className="text-center text-stone-600 text-sm mt-2">{getModeLabel(mode)}</p>
            </div>

            {/* Current User Rank */}
            {currentUserEntry && (
              <div className="mb-4 p-4 bg-stone-900 text-stone-50 rounded-lg border-2 border-stone-800">
                <div className="flex items-center justify-between">
                  <div>
                    <p className="text-xs text-stone-300 mb-1">Ваша позиція</p>
                    <p className="text-lg font-bold">{currentUserRank}. {currentUserEntry.username}</p>
                  </div>
                  <div className="text-right">
```

```
              <p className="text-xs text-stone-300 mb-1">Рейтинг</p>
              <p className="text-2xl font-bold">{currentUserEntry.elo}</p>
            </div>
          </div>
        </div>
      )}

      {/* Leaderboard */}
      {loading ? (
        <div className="bg-white border border-stone-200 rounded-lg p-8 text-center">
          <p className="text-stone-600">Завантаження...</p>
        </div>
      ) : error ? (
        <div className="bg-red-50 border border-red-200 rounded-lg p-4">
          <p className="text-red-700 text-sm">{error}</p>
          <button
            onClick={fetchLeaderboard}
            className="mt-2 text-sm text-red-700 hover:text-red-900 underline"
          >
            Спробувати ще раз
          </button>
        </div>
      ) : leaderboard.length === 0 ? (
        <div className="bg-white border border-stone-200 rounded-lg p-8 text-center">
          <p className="text-stone-600">Таблиця лідерів порожня</p>
        </div>
      ) : (
        <div className="bg-white border border-stone-200 rounded-lg overflow-hidden">
          {/* Header */}
          <div className="grid grid-cols-12 gap-2 p-3 bg-stone-100 border-b border-stone-200 text-xs font-semibold text-stone-700">
            <div className="col-span-1 text-center">#</div>
            <div className="col-span-5">Користувач</div>
            <div className="col-span-2 text-right">Рейтинг</div>
            <div className="col-span-2 text-right">Ігор</div>
            <div className="col-span-2 text-right">
              {mode === 'logicCombo' ? 'Найкращий результат' : 'Найкращий час'}
            </div>
          </div>

          {/* Rows */}
```

```
<div className="divide-y divide-stone-200">
  {leaderboard.map((entry, index) => {
    const isCurrentUser = entry.username === user?.username;
    const rankIcon = getRankIcon(entry.rank);

    return (
      <div
        key={`${entry.username}-${index}`}
        className={`grid grid-cols-12 gap-2 p-3 items-center text-sm ${
          isCurrentUser ? 'bg-stone-50 font-semibold' : ''
        }`}
      >
        <div className="col-span-1 text-center">
          {rankIcon ? (
            <span className="text-lg">{rankIcon}</span>
          ) : (
            <span className="text-stone-600">{entry.rank}</span>
          )}
        </div>
        <div className="col-span-5">
          <span className={isCurrentUser ? 'text-stone-900' : 'text-stone-700'}>
            {entry.username}
          </span>
          {isCurrentUser && (
            <span className="ml-2 text-xs text-stone-500">(Ви)</span>
          )}
        </div>
        <div className="col-span-2 text-right font-semibold text-stone-900">
          {entry.elo}
        </div>
        <div className="col-span-2 text-right text-stone-600">
          {entry.stats.wins || entry.stats.totalGames || 0}
        </div>
        <div className="col-span-2 text-right text-stone-600">
          {mode === 'logicCombo'
            ? (entry.stats.bestScore !== null ? entry.stats.bestScore : '—')
            : formatTime(entry.stats.bestTime)
          }
        </div>
      </div>
    );
```

```
                  })}
                </div>
              </div>
            )}

            {/* Action Button */}
            <div className="mt-6">
              <button
                onClick={onBack}
                className="w-full py-3 px-4 bg-stone-900 text-stone-50 rounded-lg font-sans font-semibold transition-all duration-300 hover:bg-stone-800 active:scale-95"
              >
                Повернутися до меню
              </button>
            </div>
          </div>
        </main>
      );
    }

    export default LeaderBoard;
```

ДОДАТОК Б - КОД БЕКЕНДУ СИСТЕМИ MEMORY+

Основні компоненти:

Auth.js

```
import express from 'express';
import jwt from 'jsonwebtoken';
import User from '../models/User.js';

const router = express.Router();

const generateToken = (id) => {
  return jwt.sign({ id }, process.env.JWT_SECRET, {
    expiresIn: '30d',
  });
};

router.post('/register', async (req, res) => {
  try {
    const { username, password } = req.body;

    if (!username || !password) {
      return res.status(400).json({ message: 'Будь ласка, заповніть всі поля' });
    }

    if (username.length < 3) {
      return res.status(400).json({ message: 'Ім\'я користувача повинно мати щонайменше 3 символи' });
    }

    const userExists = await User.findOne({ username });

    if (userExists) {
      return res.status(400).json({ message: 'Користувач з таким ім\'ям вже існує' });
    }

    console.log('👤 Creating new user:', username);
    const user = await User.create({
      username,
      password,
    });

    if (user) {
```

```
      console.log('✅ User created successfully in MongoDB:', user._id);
      const userObj = user.toJSON();
      res.status(201).json({
        _id: userObj._id,
        username: userObj.username,
        stats: userObj.stats,
        elo: userObj.elo,
        pairsStats: userObj.pairsStats || {},
        hintsData: userObj.hintsData,
        settings: userObj.settings,
        token: generateToken(user._id),
      });
    } else {
      res.status(400).json({ message: 'Невдалося створити користувача' });
    }
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Помилка сервера' });
  }
});

router.post('/login', async (req, res) => {
  try {
    const { username, password } = req.body;


    if (!username || !password) {
      return res.status(400).json({ message: 'Будь ласка, заповніть всі поля' });
    }

    const user = await User.findOne({ username });

    if (user && (await user.matchPassword(password))) {
      const userObj = user.toJSON();
      res.json({
        _id: userObj._id,
        username: userObj.username,
        stats: userObj.stats,
        elo: userObj.elo,
        pairsStats: userObj.pairsStats || {},
        hintsData: userObj.hintsData,
```

```
      settings: userObj.settings,
      token: generateToken(user._id),
    });
   } else {
    res.status(401).json({ message: 'Невірне ім\'я користувача або пароль' });
   }
  } catch (error) {
   console.error(error);
   res.status(500).json({ message: 'Помилка сервера' });
  }
});

export default router;
```

user.js

```
import express from 'express';
import { protect } from '../middleware/auth.js';
import User from '../models/User.js';

const router = express.Router();

router.get('/profile', protect, async (req, res) => {
  try {
   const user = await User.findById(req.user._id);
   if (!user) {
    return res.status(404).json({ message: 'Користувача не знайдено' });
   }

   const userObj = user.toJSON();
   res.json({
    _id: userObj._id,
    username: userObj.username,
    stats: userObj.stats,
    elo: userObj.elo,
    pairsStats: userObj.pairsStats || {},
    hintsData: userObj.hintsData,
    settings: userObj.settings,
   });
  } catch (error) {
   console.error(error);
   res.status(500).json({ message: 'Помилка сервера' });
  }
```

```
});


router.put('/settings', protect, async (req, res) => {
  try {
    const { animations, animationSpeed, showHints, autoSave } = req.body;

    console.log('Updating settings:', { animations, animationSpeed, showHints, autoSave });


    const user = await User.findById(req.user._id);
    if (!user) {
      return res.status(404).json({ message: 'Користувача не знайдено' });
    }

    if (!user.settings) {
      user.settings = {
        animations: true,
        animationSpeed: 'normal',
        showHints: true,
        autoSave: true,
      };
    }

    if (animations !== undefined) user.settings.animations = animations;
    if (animationSpeed !== undefined) user.settings.animationSpeed = animationSpeed;
    if (showHints !== undefined) user.settings.showHints = showHints;
    if (autoSave !== undefined) user.settings.autoSave = autoSave;

    user.markModified('settings');

    await user.save();

    console.log('Settings saved:', user.settings);

    const userObj = user.toJSON();
    res.json({
      settings: userObj.settings,
    });
  } catch (error) {
    console.error('Error updating settings:', error);
```

```
    res.status(500).json({ message: 'Помилка сервера', error: error.message });
  }
});


router.put('/stats', protect, async (req, res) => {
  try {
    console.log('📊 Updating user stats for user:', req.user._id);
    console.log('📥 Received data:', JSON.stringify(req.body, null, 2));

    const updateData = {};


    if (req.body.stats) {
      Object.keys(req.body.stats).forEach(key => {
        if (key === 'speedRecall' || key === 'logicCombo') {

          Object.keys(req.body.stats[key]).forEach(nestedKey => {
            updateData[`stats.${key}.${nestedKey}`] = req.body.stats[key][nestedKey];
          });
        } else {
          updateData[`stats.${key}`] = req.body.stats[key];
        }
      });
    }

    if (req.body.elo) {
      Object.keys(req.body.elo).forEach(key => {
        updateData[`elo.${key}`] = req.body.elo[key];
      });
    }

    if (req.body.pairsStats) {
      const user = await User.findById(req.user._id);
      if (!user) {
        return res.status(404).json({ message: 'Користувача не знайдено' });
      }


      Object.keys(req.body.pairsStats).forEach(symbol => {
        user.pairsStats.set(symbol, req.body.pairsStats[symbol]);
```

```
    });
    await user.save();
  }

  if (req.body.hintsData) {
    updateData['hintsData'] = req.body.hintsData;
  }

  if (Object.keys(updateData).length > 0) {
    console.log('💾 Saving to MongoDB with updateData:', JSON.stringify(updateData, null, 2));

    const user = await User.findByIdAndUpdate(
      req.user._id,
      { $set: updateData },
      { new: true, runValidators: true }
    );

    if (!user) {
      console.error('❌ User not found:', req.user._id);
      return res.status(404).json({ message: 'Користувача не знайдено' });
    }

    console.log('✅ User stats updated successfully');
    const userObj = user.toJSON();
    res.json({
      stats: userObj.stats,
      elo: userObj.elo,
      pairsStats: userObj.pairsStats || {},
      hintsData: userObj.hintsData,
    });
  } else if (req.body.pairsStats) {

    const user = await User.findById(req.user._id);
    const userObj = user.toJSON();
    res.json({
      pairsStats: userObj.pairsStats || {},
    });
  } else {
    res.status(400).json({ message: 'Немає даних для оновлення' });
  }
} catch (error) {
```

```
    console.error(error);
    res.status(500).json({ message: 'Помилка сервера' });
  }
});

export default router;
```

LeaderBoard.js

```
import express from 'express';
import User from '../models/User.js';

const router = express.Router();

router.get('/', async (req, res) => {
  try {
    const { mode = 'classic', limit = 50 } = req.query;

    const validModes = ['classic', 'speedRecall', 'logicCombo'];
    if (!validModes.includes(mode)) {
      return res.status(400).json({ message: 'Невірний режим гри. Доступні: classic, speedRecall, logicCombo' });
    }

    const limitNum = parseInt(limit, 10);
    if (isNaN(limitNum) || limitNum < 1 || limitNum > 100) {
      return res.status(400).json({ message: 'Ліміт повинен бути від 1 до 100' });
    }

    const users = await User.find({})
      .select('username elo stats')
      .sort({ [`elo.${mode}`]: -1 })
      .limit(limitNum)
      .lean();

    const leaderboard = users.map((user, index) => ({
      rank: index + 1,
      username: user.username,
      elo: user.elo?.[mode] || 1000,
      stats: {
        totalGames: user.stats?.totalGames || 0,
        wins: mode === 'classic'
          ? (user.stats?.winsEasy || 0) + (user.stats?.winsNormal || 0) + (user.stats?.winsHard || 0)
          : mode === 'speedRecall'
```

```
        ? user.stats?.speedRecall?.totalGames || 0
        : user.stats?.logicCombo?.totalGames || 0,
      bestTime: mode === 'classic'
        ? user.stats?.bestTimeNormal || user.stats?.bestTimeEasy || user.stats?.bestTimeHard || null
        : mode === 'speedRecall'
        ? user.stats?.speedRecall?.bestTime || null
        : null,
      bestScore: mode === 'logicCombo'
        ? user.stats?.logicCombo?.bestScore || null
        : null,
    },
  }));

  res.json({
    mode,
    leaderboard,
  });
 } catch (error) {
  console.error(error);
  res.status(500).json({ message: 'Помилка сервера' });
 }
});


router.get('/around', async (req, res) => {
 try {
  const { username, mode = 'classic', range = 5 } = req.query;

  if (!username) {
   return res.status(400).json({ message: 'Ім\'я користувача обов\'язкове' });
  }

  const validModes = ['classic', 'speedRecall', 'logicCombo'];
  if (!validModes.includes(mode)) {
   return res.status(400).json({ message: 'Невірний режим гри' });
  }

  const rangeNum = parseInt(range, 10);
  if (isNaN(rangeNum) || rangeNum < 1 || rangeNum > 20) {
   return res.status(400).json({ message: 'Діапазон повинен бути від 1 до 20' });
  }
```

```
// Find the user
const user = await User.findOne({ username }).select('username elo stats').lean();

if (!user) {
  return res.status(404).json({ message: 'Користувача не знайдено' });
}

const userElo = user.elo?.[mode] || 1000;

// Get users with higher ELO
const usersAbove = await User.find({
  [`elo.${mode}`]: { $gt: userElo },
})
  .select('username elo stats')
  .sort({ [`elo.${mode}`]: 1 })
  .limit(rangeNum)
  .lean();

// Get users with lower or equal ELO (excluding the user)
const usersBelow = await User.find({
  $or: [
    { [`elo.${mode}`]: { $lt: userElo } },
    { [`elo.${mode}`]: userElo, username: { $ne: username } },
  ],
})
  .select('username elo stats')
  .sort({ [`elo.${mode}`]: -1 })
  .limit(rangeNum)
  .lean();

// Combine and sort
const allUsers = [
  ...usersAbove.reverse(),
  user,
  ...usersBelow,
];

// Get rank of the user
const rank = await User.countDocuments({
  [`elo.${mode}`]: { $gt: userElo },
```

```
    }) + 1;

    // Format response
    const leaderboard = allUsers.map((u, index) => {
      const userRank = u.username === username ? rank : null;
      return {
        rank: userRank || null,
        username: u.username,
        elo: u.elo?.[mode] || 1000,
        isCurrentUser: u.username === username,
        stats: {
          totalGames: u.stats?.totalGames || 0,
          wins: mode === 'classic'
            ? (u.stats?.winsEasy || 0) + (u.stats?.winsNormal || 0) + (u.stats?.winsHard || 0)
            : mode === 'speedRecall'
            ? u.stats?.speedRecall?.totalGames || 0
            : u.stats?.logicCombo?.totalGames || 0,
        },
      };
    });

    res.json({
      mode,
      currentUserRank: rank,
      leaderboard,
    });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Помилка сервера' });
  }
});

export default router;
```