МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

СЕРВЕРНА ЧАСТИНА 3D-ГОЛОВОЛОМКИ «SHPERECAGE» З ЕЛЕМЕНТАМИ ФІЗИЧНОГО МОДЕЛЮВАННЯ

Виконав: студент групи 1П-22
Спеціальності
Інженерія програмного забезпечення
Ілля СОЛОВЙОВ
Керівник: Дмитро ПОДОРОШКО

Черкаси 2026

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ

Циклова комісія комп'ютерної інженерії та інформаційних технологій

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувачка циклової комісії КІ та ІТ

_____ Наталя ФАЛЬЧЕНКО
(підпис)

«_____» _____2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Соловйову Іллі Сергійовичу

1. Тема кваліфікаційної роботи Серверна частина 3D гри-головоломки «Spherecage» з елементами фізичного моделювання
Керівник роботи Подорошко Дмитро Ігорович, викладач затверджені наказом закладу фахової передвищої освіти від 06.10.2025 року No 84/1-у.

2. Строк подання студентом кваліфікаційної роботи 08.06.2026

3. Вихідні дані до кваліфікаційної роботи: прототип 3D гри-головоломки «Spherecage», Unity-клієнт з ігровим рівнем і таймером проходження, серверна частина на ASP.NET Core Web API, база даних SQLite, Entity Framework Core, а також вимоги до реєстрації користувачів, збереження результатів проходження та формування таблиці лідерів.

4. Зміст кваліфікаційної роботи: проаналізувати предметну область 3D ігор-головоломок; визначити вимоги до backend-частини гри; спроєктувати структуру API та бази даних; реалізувати серверну частину для авторизації користувачів, збереження результатів, обліку прогресу гравця й отримання таблиці лідерів; провести функціональне та змодельоване навантажувальне тестування.

5. Дата видачі завдання 15.09.2025 р.

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання
	Вступ		
	Аналіз предметної області та існуючих 3D ігор-головоломок		
	Проєктування клієнт-серверної архітектури прототипу «Spherecage»		
	Реалізація та тестування backend-частини прототипу		
	Висновки		
	Оформлення кваліфікаційної роботи (чистовий варіант)		
	Перевірка кваліфікаційної роботи на наявність ознак плагіату (за 10 днів до захисту)		
	Подання кваліфікаційної роботи на затвердження завідувачу циклової комісії (за 7 днів до захисту)		

Студент _____ Ілля СОЛОВЙОВ

(підпис)

Керівник роботи _____ Дмитро ПОДОРОШКО

(підпис)

АНОТАЦІЯ

У кваліфікаційній роботі розглянуто розробку серверної частини 3D гри-головоломки «Spherecage». Проєкт є прототипом клієнт-серверної гри, у якій користувач проходить рівень із фізичною взаємодією кулі та лабіринту, а результат проходження зберігається на сервері та використовується для формування таблиці лідерів.

У межах роботи проаналізовано предметну область 3D ігор-головоломок, визначено роль backend-складової у невеликому ігровому застосунку, спроєктовано структуру даних і API для взаємодії Unity-клієнта з сервером. Серверну частину реалізовано з використанням ASP.NET Core Web API, Entity e та SQLite. Реалізовано реєстрацію й авторизацію користувачів, збереження результатів проходження рівня, отримання таблиці лідерів та зберігання базового прогресу гравця.

Проведено функціональне тестування основних API-запитів, перевірено сценарії взаємодії клієнта і сервера, а також змодельовано просте навантажувальне тестування запиту отримання таблиці лідерів. Практичне значення роботи полягає у створенні працездатної backend-основи для розважального ігрового контенту, яка може бути використана для подальшого розвитку гри, розширення ігрового жанру та вдосконалення унікальної механіки проходження рівнів.

ЗМІСТ

ВСТУП

У сучасних умовах розвитку інформаційних технологій комп’ютерні ігри вже давно перестали бути лише засобом розваги. Сьогодні ігрові застосунки є повноцінними програмними системами, у яких поєднуються елементи інтерфейсу користувача, програмної логіки, фізичного моделювання, обробки подій та збереження даних. Особливо це стосується 3D ігор, у яких важливе значення має не лише візуальне представлення об’єктів, а й організація внутрішньої логіки системи та взаємодія окремих компонентів програмного продукту.

Одним із важливих напрямів розвитку сучасних ігрових застосунків є використання клієнт-серверної архітектури. Такий підхід дає можливість відокремити клієнтську частину, яка відповідає за відображення гри та взаємодію з користувачем, від серверної частини, яка забезпечує обробку запитів, збереження результатів, роботу з базою даних та підтримку спільної логіки системи. Навіть у відносно простих прототипах або демонстраційних ігрових проєктах наявність серверної частини дозволяє реалізувати функції, які неможливо або недоцільно повністю покладати лише на клієнт: реєстрацію користувачів, облік результатів проходження, формування таблиці лідерів, централізоване збереження даних.

Актуальність теми зумовлена тим, що при розробці програмних продуктів важливо не лише створити візуально працездатний застосунок, а й побудувати структуровану серверну частину, яка забезпечує коректну обробку даних та взаємодію між компонентами системи. Для 3D гри-головоломки «Spherecage» така серверна частина є важливою складовою, оскільки саме вона відповідає за збереження інформації про користувачів, результати проходження рівня та надання даних для таблиці рекордів. У межах спільного проєкту розроблення гри окремий розгляд backend-складової дозволяє зосередитися на задачах серверної архітектури та інтеграції ігрового клієнта з Web API, що відповідає вимозі індивідуального інженерного фокусу під час спільної розробки одного програмного продукту.

Метою кваліфікаційної роботи є розробка серверної частини 3D гри-головоломки «Spherecage», яка забезпечує реєстрацію та авторизацію користувачів, збереження результатів проходження рівня, взаємодію з клієнтською частиною гри та формування таблиці лідерів.

Об’єктом дослідження є клієнт-серверні програмні системи у сфері ігрових застосунків.

Предметом дослідження є методи та засоби реалізації серверної частини 3D гри-головоломки з використанням ASP.NET Core Web API, Entity Framework Core та SQLite.

Практичне значення одержаних результатів полягає у створенні працездатної серверної частини для прототипу ігрового проєкту, яка може бути використана як основа для подальшого розвитку гри «Spherecage», розширення її функціональних можливостей та інтеграції з більш складними клієнтськими сценаріями. Також результати роботи можуть бути використані як приклад реалізації backend-складової для невеликих клієнт-серверних застосунків і прототипів у галузі інженерії програмного забезпечення.

Для досягнення поставленої мети у роботі визначено такі завдання:

– проаналізувати предметну область 3D ігор-головоломок та визначити роль серверної частини у збереженні користувацьких даних і результатів проходження;

– розглянути наявні рішення та механіки, близькі до ідеї гри «Spherecage», зокрема ігри з керуванням кулею та проходженням просторового лабіринту;

– спроєктувати клієнт-серверну архітектуру прототипу з розподілом відповідальності між Unity-клієнтом, Web API та базою даних SQLite;

– розробити структуру бази даних для зберігання користувачів, результатів проходження рівня та базового прогресу гравця;

– реалізувати backend-частину на ASP.NET Core Web API з використанням Entity Framework Core для доступу до SQLite;

– забезпечити обмін даними між Unity-клієнтом і сервером через HTTP-запити у форматі JSON;

– провести тестування основних сценаріїв: реєстрації, авторизації, збереження результату, отримання таблиці лідерів та перевірки роботи API під простим модельованим навантаженням.

У межах спільної розробки гри індивідуальний фокус даної кваліфікаційної роботи зосереджено на backend-складовій: проєктуванні серверної архітектури, реалізації API, організації збереження даних у SQLite та перевірці взаємодії сервера з Unity-клієнтом. Ігрова сцена, фізична механіка та візуальна частина розглядаються у роботі як клієнтський контекст, необхідний для пояснення взаємодії з сервером.

РОЗДІЛ 1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Характеристика предметної області

Предметною областю даної кваліфікаційної роботи є розробка програмних систем у сфері комп'ютерних ігор, зокрема 3D ігор-головоломок, що використовують клієнт-серверну архітектуру для обробки та збереження даних. Сучасні ігрові застосунки поєднують у собі декілька взаємопов'язаних компонентів: клієнтську частину, що відповідає за візуальне відображення та взаємодію з користувачем, і серверну частину, що забезпечує збереження даних, обробку запитів та централізовану логіку роботи системи.

У межах даної роботи розглядається 3D гра-головоломка «Spherecage». Концепція гри полягає в тому, що користувач керує ігровим об'єктом у межах лабіринту та повинен дістатися до фінішної точки. Основна увага в самій грі приділяється простоті механіки, фізичній взаємодії об'єктів та наочності проходження рівня. Однак для забезпечення повноцінної роботи програмного продукту важливою є не лише клієнтська частина, а й серверна інфраструктура, яка дозволяє працювати з даними користувачів та результатами гри.

У такій системі можна виділити декілька основних сутностей. Першою сутністю є користувач, який запускає гру, вводить власне ім'я та взаємодіє з ігровим середовищем. Другою сутністю є клієнтська частина, реалізована у середовищі Unity, яка відповідає за відображення рівня, керування, інтерфейс та надсилання запитів на сервер. Третьою сутністю є серверна частина, реалізована як Web API, яка приймає запити клієнта, здійснює авторизацію або реєстрацію користувача, приймає результати проходження рівня та формує таблицю лідерів. Четвертою сутністю є база даних, у якій зберігаються користувацькі записи та результати проходження.

Основною проблемою в межах цієї предметної області є необхідність організувати надійне збереження ігрових даних. Якщо гра працює виключно як локальний клієнтський застосунок, то результати користувача або не зберігаються взагалі, або зберігаються лише на одному пристрої без можливості

централізованого обліку. У такому випадку неможливо побудувати повноцінну таблицю рекордів, реалізувати ідентифікацію користувачів або забезпечити єдиний механізм доступу до результатів гри. Саме тому доцільним є використання серверної частини, яка бере на себе задачі збереження, структурування та надання даних клієнтському застосунку.

Для гри «Spherecage» серверна частина виконує декілька важливих функцій. По-перше, вона забезпечує реєстрацію та авторизацію користувача, що дозволяє ідентифікувати гравця в системі. По-друге, сервер приймає результати проходження рівня, зокрема час, за який користувач досяг фінішу. По-третє, збережені результати використовуються для формування таблиці лідерів, яка може відображатися у клієнтській частині гри. Таким чином, сервер стає не допоміжним, а повноцінним компонентом програмної системи.

З точки зору програмної інженерії, дана предметна область є показовою для реалізації клієнт-серверної взаємодії у невеликому ігровому проєкті. Вона дозволяє об’єднати практичні аспекти розробки ігрової логіки з питаннями архітектури, проєктування API, збереження даних та інтеграції різних програмних компонентів. Це робить тему роботи актуальною не лише в контексті розробки конкретної гри, а й у межах підготовки фахівця з інженерії програмного забезпечення.

Наприкінці підрозділу можна зробити висновок, що предметна область роботи охоплює не лише саму 3D гру як інтерактивний застосунок, а й серверну частину як ключовий інструмент організації користувацьких даних, результатів проходження та взаємодії між компонентами системи.

.2 Аналіз наявних рішень

У процесі розробки програмних продуктів важливим етапом є аналіз вже існуючих рішень, які належать до тієї ж або суміжної предметної області. Такий аналіз дозволяє визначити сильні та слабкі сторони аналогів, виявити типові підходи до реалізації необхідної функціональності, а також обґрунтувати доцільність власних інженерних рішень. У межах даної роботи аналіз наявних

рішень доцільно розглядати у двох напрямах: як аналіз ігрових застосунків із подібною механікою та як аналіз програмних рішень, що використовують серверну частину для збереження користувацьких даних і результатів.

Серед ігор-головоломок досить поширеними є проєкти, у яких гравець взаємодіє з фізичним середовищем, долає перешкоди, проходить лабіринт або виконує послідовність дій для досягнення мети. Такі ігри приваблюють користувачів простими правилами, але водночас потребують продуманої реалізації механіки руху, взаємодії з рівнем та логіки завершення етапу. Більшість подібних невеликих проєктів мають локальний характер і не використовують окрему серверну частину. У таких випадках результати проходження або не зберігаються взагалі, або зберігаються лише локально, що обмежує можливості подальшого розвитку системи.

Інший клас рішень становлять клієнт-серверні застосунки, у яких сервер використовується для централізованого збереження даних. У сфері ігрових продуктів такий підхід дозволяє реалізувати облік користувачів, фіксацію результатів, таблиці рекордів та інші функції, пов’язані з багатокористувацьким або хоча б централізованим використанням даних. Навіть якщо гра є відносно простою з точки зору візуальної частини та механіки, наявність серверної складової значно підвищує її структурованість і практичну цінність як програмного продукту.

Аналіз наявних рішень показує, що найпростішим підходом є повністю локальна реалізація, коли вся логіка та збереження даних розташовані на стороні клієнта. Перевагами такого підходу є простота реалізації, відсутність потреби в окремому сервері та мінімальні вимоги до інфраструктури. Однак недоліки такого варіанта є суттєвими: відсутня централізована база даних, неможливо реалізувати повноцінний облік користувачів, важко забезпечити спільну таблицю результатів, а також ускладнюється подальше розширення функціоналу.

Більш гнучким рішенням є використання клієнт-серверної архітектури. У такому випадку клієнтська частина відповідає за безпосередню взаємодію з користувачем, а серверна – за обробку запитів і роботу з даними. Основною

перевагою цього підходу є централізоване збереження результатів та логічне розмежування відповідальності між компонентами системи. Крім того, така архітектура є більш показовою з точки зору інженерії програмного забезпечення, оскільки дозволяє окремо проєктувати API, базу даних та сценарії взаємодії компонентів.

Для реалізації серверної частини навчальних або демонстраційних проєктів досить часто використовують Web API, побудовані на сучасних фреймворках. Такий підхід є зручним завдяки простій організації обміну даними у форматі JSON, можливості використання стандартних HTTP-запитів, а також чіткій структурі взаємодії між клієнтом і сервером. Порівняно з більш складними архітектурними рішеннями, Web API є достатньо простим для реалізації у межах навчального проєкту та водночас повністю покриває потреби збереження ігрових результатів, авторизації та отримання таблиці лідерів.

Якщо розглядати існуючі рішення з точки зору функціональних можливостей, то можна виділити кілька типових варіантів. Перший варіант – локальна гра без авторизації та без збереження даних. Другий варіант – гра із локальним збереженням прогресу. Третій варіант – клієнт-серверна система, у якій реалізовано роботу з користувачами, результатами та таблицею рекордів. Саме третій варіант є найбільш доцільним для реалізації в межах роботи над серверною частиною гри «Spherecage», оскільки він дозволяє продемонструвати не лише роботу ігрового клієнта, а й побудову окремої серверної підсистеми.

Порівняння наявних підходів дає підстави зробити висновок, що для розроблюваного програмного продукту найбільш виправданим є використання клієнт-серверної архітектури з окремим API та базою даних. Такий вибір зумовлений необхідністю забезпечити реєстрацію або ідентифікацію користувача, збереження результатів проходження рівня, а також можливість отримання та відображення таблиці лідерів. На відміну від повністю локальних рішень, обраний підхід забезпечує кращу структурованість системи, розширюваність та більш повне розкриття інженерного аспекту розробки.

Отже, аналіз наявних рішень показав, що більшість простих ігрових проєктів орієнтовані лише на клієнтську частину та не враховують потребу в централізованому збереженні даних. Для гри «Spherecage» доцільно використати більш системний підхід із виділенням серверної частини як окремого компонента, що забезпечує реєстрацію користувачів, збереження результатів та формування таблиці лідерів. Саме такий підхід і буде реалізовано в межах подальших розділів роботи. Порівняння ігрових рішень наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння ігрових рішень, близьких до ідеї Spherecage

Рішення	Особливості	Висновок для Spherecage
	Керування кулею та проходження просторових перешкод; результат залежить від проходження рівнів.	Підтверджує цікавість механіки руху кулі як основи головоломки.
	Рух кулі або персонажа у замкненому просторі; орієнтація на швидкість і проходження рівнів.	Показує важливість зрозумілого керування та коротких рівнів.
	Проведення кулі через рівень із перешкодами; результат пов’язаний із часом і проходженням.	Дає орієнтир для поєднання фізики, лабіринту та таймера.
	Керування кулею у 3D лабіринті; час проходження зберігається через Web API та	Прототип поєднує ігрову механіку з серверним збереженням даних.

Порівняння показує, що ідея гри «Spherecage» спирається на зрозумілу для жанру механіку керування кулею, але доповнює її backend-складовою: збереженням користувача, результату проходження та формуванням таблиці лідерів. Саме ця серверна частина є основним інженерним фокусом даної роботи.

1.3 Постановка завдання

На основі проведеного аналізу предметної області та наявних рішень можна сформулювати завдання кваліфікаційної роботи. У межах спільного проєкту «Spherecage» розроблювана система являє собою 3D гру-головоломку з клієнтською частиною, реалізованою в середовищі Unity, та серверною частиною, реалізованою у вигляді Web API. У даній роботі основна увага

приділяється саме серверній частині системи, яка забезпечує обробку запитів від клієнта, збереження користувацьких даних та результатів проходження гри.

Цільове призначення системи полягає в забезпеченні взаємодії між клієнтською частиною гри та сервером для виконання базових сценаріїв роботи. До таких сценаріїв належать введення імені користувача, реєстрація або авторизація гравця, надсилання результату після завершення рівня, збереження цього результату в базі даних та отримання таблиці лідерів для подальшого відображення у клієнтському інтерфейсі.

Крім функціональних вимог, для системи важливими є нефункціональні вимоги. Серверна частина повинна бути достатньо простою для розгортання та використання в межах навчального проєкту. Вона повинна мати зрозумілу структуру, бути зручною для інтеграції з Unity-клієнтом та забезпечувати коректну обробку основних запитів без надмірної складності архітектури. Обмін даними має бути реалізований із використанням стандартних підходів до побудови Web API. Для зберігання даних доцільно використати легку реляційну базу даних, яка не потребує складного адміністрування. Також важливою вимогою є достатня швидкодія системи для обробки запитів реєстрації, авторизації, збереження результатів і отримання таблиці лідерів.

До системних обмежень, які впливають на подальші проєктні рішення, належать навчальний характер проєкту, обмежений час розробки та використання одного демонстраційного рівня в клієнтській частині гри. Серверна частина орієнтована на локальне або спрощене середовище виконання, а механізм авторизації реалізується у базовому вигляді, достатньому для демонстрації клієнт-серверної взаємодії. Також обмеженням є те, що система створюється як частина спільного програмного продукту, у якому клієнтська частина розглядається як зовнішній інтеграційний компонент.

Отже, у межах кваліфікаційної роботи необхідно розробити серверну частину гри «Spherecage», яка забезпечить ідентифікацію користувача, збереження ігрових результатів та формування таблиці лідерів, а також дозволить клієнтській частині гри взаємодіяти із сервером через Web API. Сформульовані

вимоги та обмеження є основою для подальшого проєктування архітектури системи, структури даних та програмної реалізації серверної частини.

У першому розділі було розглянуто предметну область розробки 3D гри-головоломки з використанням клієнт-серверної архітектури. Визначено, що для сучасних ігрових застосунків важливим є не лише створення клієнтської частини, яка забезпечує візуальне відображення гри та взаємодію з користувачем, а й реалізація серверної складової, що відповідає за обробку запитів і збереження даних.

У ході аналізу наявних рішень було встановлено, що більшість простих ігрових проєктів реалізуються як локальні застосунки без централізованого збереження даних. Такий підхід спрощує розробку, однак обмежує функціональні можливості системи, зокрема в частині ідентифікації користувачів, збереження результатів проходження та формування таблиці лідерів. Водночас використання клієнт-серверної архітектури дає можливість організувати обробку та зберігання даних у більш структурований спосіб.

На основі проведеного аналізу було сформульовано основні вимоги до серверної частини гри «Spherecage». Визначено, що система повинна забезпечувати реєстрацію або авторизацію користувача, приймання та збереження результатів проходження рівня, а також надання таблиці лідерів клієнтській частині гри. Крім того, встановлено нефункціональні вимоги до системи, серед яких простота інтеграції з Unity, використання стандартного формату обміну даними та достатня продуктивність у межах навчального проєкту.

Отже, результати першого розділу дозволили обґрунтувати доцільність розробки окремої серверної частини для гри «Spherecage» та сформували основу для подальшого проєктування архітектури системи, структури даних і програмної реалізації.

РОЗДІЛ 2 ПРОЄКТУВАННЯ BACKEND-ЧАСТИНИ ГРИ «SPHERECAGE»

2.1 Загальна архітектура системи

Після визначення загальної архітектури системи наступним етапом є проєктування структури даних та програмного інтерфейсу взаємодії між клієнтською і серверною частинами. У межах проєкту «Spherecage» серверна частина має виконувати обмежений, але чітко визначений набір задач: ідентифікацію користувача, приймання результатів проходження рівня та надання таблиці лідерів. Відповідно до цього було спроєктовано просту структуру даних, яка не перевантажує систему зайвими сутностями, але дозволяє реалізувати весь необхідний функціонал.

У поточній реалізації основу структури даних складають дві головні сутності: користувач та результат гри. Сутність користувача використовується для збереження мінімального набору відомостей, необхідних для реєстрації та подальшої ідентифікації гравця. До її складу входять ідентифікатор користувача, ім’я користувача та хеш пароля. Ідентифікатор є унікальним ключем запису, ім’я використовується під час входу та відображення у таблиці лідерів, а пароль у межах навчального проєкту використовується в спрощеному вигляді для базової авторизації.

Друга сутність – результат гри – призначена для збереження інформації про проходження рівня. Вона містить ідентифікатор запису, ідентифікатор користувача, номер рівня, час проходження та дату збереження результату. Саме ця сутність є основою для побудови таблиці лідерів, оскільки дозволяє пов’язати конкретного користувача з конкретним результатом і відсортувати записи за часом проходження.

Зв’язок між цими сутностями є простим: один користувач може мати декілька результатів проходження. Такий підхід дозволяє зберігати не лише один запис, а історію результатів, що є більш гнучким рішенням навіть для невеликого навчального проєкту. Для реалізації доступу до даних у серверній частині використовується Entity Framework Core, який дає змогу працювати з таблицями

бази даних через моделі та контекст даних без написання великої кількості ручних SQL-запитів.

Після проєктування структури даних було визначено набір API-запитів, необхідних для взаємодії з клієнтом Unity. У межах системи реалізовано чотири основні маршрути. Перший маршрут призначений для реєстрації користувача. Він приймає ім’я користувача та пароль і створює новий запис у базі даних, якщо такий користувач ще не існує. Другий маршрут використовується для авторизації. Він перевіряє, чи є в системі користувач із переданими обліковими даними, та у разі успіху повертає клієнту ідентифікатор користувача.

Третій маршрут використовується для надсилання результату гри. Після завершення рівня Unity-клієнт формує запит, який містить ідентифікатор користувача, номер рівня та час проходження. Сервер приймає ці дані, створює новий запис у таблиці результатів та зберігає його в базі даних. Четвертий маршрут відповідає за отримання таблиці лідерів. Він повертає список найкращих результатів, відсортованих за часом проходження, разом з іменами користувачів.

Для передавання даних між клієнтом і сервером використовується формат JSON. Такий вибір є природним для Web API, а також зручним для Unity, оскільки дозволяє легко серіалізувати та десеріалізувати об’єкти. Наприклад, при надсиланні результату клієнт формує JSON-об’єкт із полями userId, levelNumber та completionTime. У відповідь сервер повертає повідомлення про успішне збереження. Аналогічно при запиті таблиці лідерів сервер повертає масив об’єктів, кожен із яких містить ім’я користувача, номер рівня, час проходження та дату запису результату.

З точки зору клієнтської частини така організація API є зручною, оскільки кожен сценарій має окремий маршрут і чітко визначений формат даних. Це спрощує інтеграцію з Unity та зменшує ризик помилок під час обміну інформацією між частинами системи. У межах реалізації гри звернення до API здійснюються через окремий клієнтський компонент, який формує HTTP-запити та обробляє відповіді сервера.

Отже, спроєктована структура даних та набір API-запитів є достатніми для реалізації основного функціоналу серверної частини гри «Spherecage». Вони забезпечують збереження інформації про користувачів, приймання результатів проходження рівня та формування таблиці лідерів, а також створюють основу для подальшого розширення системи в разі потреби.

2.2 Проєктування структури даних та API

Після того як була визначена загальна архітектура системи, потрібно було вирішити дві практичні речі: які саме дані має зберігати сервер і яким чином Unity-клієнт буде з ним взаємодіяти. Оскільки гра «Spherecage» у межах цієї роботи є невеликим клієнт-серверним проєктом, надто складну структуру даних створювати не було сенсу. Навпаки, важливо було залишити тільки ті сутності, без яких основний функціонал гри не працював би взагалі.

У поточній реалізації серверна частина працює з двома основними сутностями: користувачем і результатом проходження. Сутність користувача потрібна для того, щоб сервер міг відрізняти одного гравця від іншого. У ній зберігаються ідентифікатор користувача, його ім’я та пароль. Ідентифікатор є унікальним і використовується як основний ключ. Ім’я використовується під час входу та надалі відображається в таблиці лідерів. Пароль у межах навчального проєкту потрібен для базової авторизації, щоб клієнт не працював просто з анонімними записами.

Друга сутність – це результат проходження рівня. Вона містить ідентифікатор запису, ідентифікатор користувача, номер рівня, час проходження та дату збереження результату. Саме ця таблиця є основою для подальшого формування таблиці лідерів. Для поточного проєкту цього цілком достатньо, оскільки в грі поки реалізовано один основний рівень і головний показник, який потрібно зберігати, – це час його проходження.

Між цими двома сутностями існує простий зв’язок: один користувач може мати декілька результатів. Такий підхід є логічним, оскільки один і той самий гравець може проходити рівень декілька разів, а система при цьому повинна

зберігати його спроби. Навіть у спрощеній реалізації це виглядає більш правильно, ніж зберігати для кожного користувача лише один єдиний результат.

Під час проєктування структури даних свідомо не додавались додаткові таблиці чи складні зв’язки. Наприклад, можна було окремо зберігати прогрес, налаштування користувача або службову історію дій. Але для навчального проєкту це лише ускладнило б реалізацію та відволікло б увагу від головного – побудови серверної частини, яка реально працює разом із клієнтом Unity. Саме тому структура бази даних залишилась компактною і зосередженою на основному функціоналі.

Після цього було визначено набір API-запитів, через які клієнтська частина звертається до сервера. Для поточної реалізації достатньо чотирьох основних маршрутів. Перший використовується для реєстрації користувача. Якщо гравець вводить ім’я, якого ще немає в системі, сервер створює новий запис і повертає відповідь клієнту. Другий маршрут відповідає за авторизацію. Якщо користувач уже існує, клієнт може отримати його дані та працювати далі вже з конкретним

Третій маршрут потрібен для надсилання результату після завершення рівня. У цей момент Unity-клієнт передає на сервер ідентифікатор користувача, номер рівня та час проходження. Сервер приймає ці дані, формує новий запис у таблиці результатів і зберігає його в базі. Четвертий маршрут використовується для отримання таблиці лідерів. За цим запитом сервер повертає список результатів, відсортованих за часом проходження, щоб клієнт міг відобразити їх у вигляді leaderboard.

Для обміну даними був використаний формат JSON. Це стандартне й зручне рішення для Web API, а для Unity воно підходить ще й тому, що такі об’єкти легко серіалізувати й обробляти на стороні клієнта. Наприклад, при відправленні результату формується об’єкт із полями userId, levelNumber та completionTime. У відповідь сервер повертає повідомлення про успішне збереження. Якщо ж клієнт запитує таблицю лідерів, сервер повертає вже список

записів, у якому містяться ім’я користувача, рівень, час проходження та дата збереження результату.

З практичної точки зору така організація API виявилась зручною ще й тому, що кожен основний сценарій гри має окремий маршрут. Через це клієнтська частина не перевантажена зайвою логікою, а серверна частина залишається зрозумілою за структурою. Для навчального проєкту це важливо, оскільки система повинна бути не лише працездатною, а й достатньо простою для пояснення, тестування та подальшого розширення.

Отже, спроєктована структура даних та API повністю покривають потреби поточної версії гри «Spherecage». Вони дозволяють реєструвати користувачів, зберігати результати проходження та отримувати таблицю лідерів, а також створюють основу для подальшого розвитку серверної частини без необхідності повного перепроєктування системи.

2.3 Обґрунтування вибору технологій

Під час розробки серверної частини гри «Spherecage» потрібно було обрати такий набір технологій, який дозволив би швидко реалізувати основний функціонал, не ускладнюючи проєкт зайвими рішеннями. Оскільки в межах роботи сервер повинен обробляти запити від Unity-клієнта, зберігати дані користувачів і результати проходження, головними критеріями вибору стали простота реалізації, зручність інтеграції та достатність для навчального клієнт-серверного проєкту.

Як основу серверної частини було обрано ASP.NET Core Web API. Це рішення є зручним для побудови HTTP-сервісів, оскільки дозволяє швидко створювати окремі маршрути для різних сценаріїв роботи та логічно розділяти застосунок на контролери, моделі й компоненти доступу до даних. Для гри «Spherecage» це виявилося доречним, оскільки всі необхідні дії – реєстрація, авторизація, збереження результату та отримання таблиці лідерів – природно оформлюються саме у вигляді окремих API-запитів. Крім того, ASP.NET Core

добре документований, має зрозумілу структуру проєкту й підходить для демонстрації принципів побудови серверної частини в дипломній роботі.

Для зберігання даних було використано SQLite. У межах поточного проєкту це найбільш практичний варіант, оскільки він не потребує окремого серверу баз даних і легко підходить для локального розгортання. Гра зберігає лише користувачів і результати проходження, тому використовувати складнішу СУБД не було необхідності. SQLite дозволяє організувати повноцінне збереження даних без зайвого навантаження на проєкт, що особливо важливо в умовах обмеженого часу розробки.

Для взаємодії з базою даних було обрано Entity Framework Core. Його використання дозволило працювати з таблицями через моделі та контекст даних, а не писати більшість операцій вручну на SQL. У практичному плані це спростило реалізацію додавання нових користувачів, пошуку записів, збереження результатів і вибірки таблиці лідерів. Для навчального проєкту це також є плюсом, оскільки структура коду стає більш зрозумілою, а серверна частина – простішою для пояснення та підтримки.

Клієнтська частина гри реалізована в середовищі Unity. Хоча основний фокус даної роботи зроблено саме на backend-складовій, вибір Unity також вплинув на серверні рішення. Unity зручно працює з HTTP-запитами та JSON, тому серверна частина відразу проєктувалася так, щоб клієнт міг без зайвих труднощів надсилати дані про користувача, результати проходження та отримувати таблицю лідерів. Саме тому вибір Web API і JSON як основного формату обміну даними виявився найбільш природним.

Окремо варто зазначити, що при виборі технологій враховувалась не лише їх популярність, а й відповідність самому масштабу проєкту. Для невеликої навчальної гри не було сенсу будувати складну багаторівневу архітектуру або використовувати інструменти, які перевищують реальні потреби системи. Навпаки, доцільніше було обрати компактний і зрозумілий стек, який забезпечує працездатність продукту та дозволяє показати повноцінну взаємодію між клієнтом, сервером і базою даних.

Отже, вибір ASP.NET Core Web API, SQLite, Entity Framework Core та Unity є обґрунтованим з точки зору цілей проєкту. Саме цей стек дозволив реалізувати серверну частину гри «Spherecage» у зрозумілій формі, забезпечити збереження даних і побудувати стабільну взаємодію між усіма основними компонентами системи.

2.4 Структура серверної частини проєкту

Після вибору основних технологій та визначення загальної архітектури системи наступним кроком стало формування зрозумілої структури серверної частини проєкту. Для навчального клієнт-серверного застосунку це має велике значення, оскільки навіть відносно невеликий backend-модуль повинен бути організований так, щоб окремі частини системи не змішувалися між собою, а логіка залишалася зрозумілою під час розробки, налагодження та подальшого опису в пояснювальній записці.

Серверна частина гри «Spherecage» була реалізована як окремий проєкт на API. Такий підхід дозволив відразу відокремити її від клієнтської частини, яка працює в Unity. У результаті гра і сервер запускаються незалежно один від одного, але взаємодіють через HTTP-запити. Це не лише зручно з точки зору розробки, а й відповідає самій ідеї клієнт-серверної архітектури, де кожен компонент виконує свою чітко визначену роль.

У структурі серверного проєкту можна виділити кілька основних логічних частин. Першою з них є моделі даних. Саме вони описують сутності, з якими працює система, тобто користувача та результат проходження рівня. Моделі визначають, які саме поля зберігаються у базі даних, як вони пов’язані між собою та які об’єкти використовуються під час роботи серверної логіки.

Наступною важливою частиною є компонент доступу до даних, представлений контекстом бази даних. Через нього сервер взаємодіє з таблицями SQLite, додає нові записи, виконує пошук, отримує результати та формує вибірки для таблиці лідерів. Використання окремого контексту дозволило не змішувати

логіку роботи з базою із логікою обробки HTTP-запитів, що зробило код більш упорядкованим.

Окреме місце в структурі проєкту займають контролери. Саме вони приймають запити від Unity-клієнта та запускають потрібний сценарій роботи. У поточній реалізації один контролер відповідає за реєстрацію та авторизацію користувачів, інший – за збереження результатів гри, а ще один – за повернення таблиці лідерів. Такий розподіл виявився зручним, оскільки кожен блок відповідає за свою окрему задачу і не перевантажується зайвим функціоналом.

Важливу роль у серверній частині також відіграє файл конфігурації запуску застосунку. У ньому налаштовується підключення до бази даних, реєстрація необхідних сервісів, підтримка контролерів та допоміжних інструментів, зокрема Swagger. Саме через цей компонент сервер отримує готове середовище для роботи, у якому вже підключені база даних, маршрути та інші необхідні частини системи.

З практичної точки зору така структура проєкту виявилася вдалою ще й тому, що вона спростила налагодження. Якщо виникала проблема із входом користувача, її можна було шукати в логіці авторизації. Якщо не зберігався результат – перевірялась відповідна модель, маршрут і запис у базу даних. Якщо некоректно працював leaderboard – перевірялась вибірка результатів і формат відповіді сервера. Завдяки цьому серверна частина залишалася не лише працездатною, а й зручною для поетапної перевірки.

Ще однією перевагою такої структури є можливість подальшого розширення. Навіть у поточному вигляді проєкт можна доповнити новими маршрутами, окремими сервісами або додатковими сутностями без повного переписування вже існуючого коду. Наприклад, у майбутньому до серверної частини можна додати збереження прогресу, роботу з кількома рівнями, розширену статистику або окремі ролі користувачів.

Отже, структура серверної частини проєкту «Spherecage» була побудована за принципом логічного розділення основних компонентів: моделей, роботи з базою даних, контролерів і конфігурації застосунку. Такий підхід дозволив

зберегти код зрозумілим, спростив інтеграцію з Unity-клієнтом і створив основу для подальшого розвитку серверного модуля.

2.5 Сценарії взаємодії клієнта і сервера

Для того щоб серверна частина гри «Spherecage» працювала не ізольовано, а як повноцінний компонент системи, потрібно було чітко визначити, у яких саме моментах клієнт Unity звертається до сервера і які дані при цьому передаються. На практиці вся взаємодія між клієнтом і сервером у поточній версії проєкту будується навколо кількох простих, але важливих сценаріїв.

Перший сценарій пов’язаний із початком роботи користувача з грою. Після запуску застосунку гравець вводить своє ім’я у стартовому меню. Після цього клієнтська частина формує запит до серверної частини для реєстрації або авторизації. Якщо такого користувача в системі ще немає, сервер створює новий запис у базі даних. Якщо ж такий користувач уже існує, сервер виконує авторизацію і повертає клієнту дані, необхідні для подальшої роботи. У результаті Unity-клієнт отримує ідентифікатор користувача, з яким працює надалі.

Другий сценарій відбувається вже під час ігрового процесу, хоча сам сервер у цей момент напряму в логіку гри не втручається. Основна механіка проходження рівня, рух гравця, взаємодія з лабіринтом і визначення моменту завершення етапу залишаються на стороні Unity. Серверна частина підключається лише тоді, коли рівень завершено і потрібно зафіксувати результат проходження.

Третій сценарій починається в момент досягнення фінішу. Після завершення рівня клієнт отримує підсумкове значення часу проходження, а потім формує запит до сервера. У цьому запиті передається ідентифікатор користувача, номер рівня та час проходження. Сервер приймає ці дані, створює новий запис у таблиці результатів і зберігає його в базі даних. Саме цей сценарій є ключовим для backend-частини, оскільки він перетворює локальний результат гри на збережені дані, доступні системі надалі.

Четвертий сценарій пов’язаний із переглядом таблиці лідерів. Коли користувач відкриває leaderboard, клієнтська частина надсилає окремий запит до серверної частини. У відповідь сервер виконує вибірку результатів із бази даних, сортує їх за часом проходження і повертає вже готовий список. Unity-клієнт приймає ці дані та відображає їх у зручному для користувача вигляді. Таким чином, сервер відповідає не лише за збереження інформації, а й за її підготовку до показу в клієнтському інтерфейсі.

Усі ці сценарії разом формують замкнений цикл взаємодії між компонентами системи. Користувач заходить у гру, вводить ім’я, сервер повертає дані користувача, після проходження рівня клієнт надсилає результат, сервер його зберігає, а потім за запитом повертає оновлену таблицю лідерів. На практиці це означає, що навіть за відносно простої механіки самої гри система вже працює як повноцінний клієнт-серверний застосунок.

З практичної точки зору така організація сценаріїв виявилася вдалою, тому що вона не перевантажує ні клієнтську, ні серверну частину. Unity займається лише тим, що стосується ігрового процесу та інтерфейсу, а сервер бере на себе роботу з користувачами, результатами і таблицею лідерів. Саме таке розмежування і було однією з основних цілей під час побудови архітектури проєкту.

Отже, визначені сценарії взаємодії клієнта і сервера забезпечують узгоджену роботу всіх компонентів системи. Вони дозволяють організувати ідентифікацію користувача, збереження ігрових результатів та отримання таблиці ідерів у простій і зрозумілій формі, що є достатнім для поточної версії гри

2.6 Версія API та документація запитів

У поточній реалізації прототипу використано першу функціональну версію API, яка охоплює базові сценарії роботи гри: авторизацію користувача, збереження результату, отримання таблиці лідерів і роботу з прогресом. У маршрутах контролерів застосовано схему /api/[controller], тому версія API не

винесена в URL окремим сегментом. Для прототипу цього достатньо, оскільки клієнт і сервер розробляються як одна узгоджена система. У разі подальшого розвитку доцільно перейти до явного маршрутування на зразок /api/v1/Auth, /api/v1/Results та /api/v1/Leaderboard, щоб нові версії API можна було додавати без порушення роботи наявного Unity-клієнта.

Для перевірки та демонстрації серверних запитів у проєкті підключено Swagger/OpenAPI через пакет Swashbuckle.AspNetCore. У середовищі розробки це дозволяє переглядати доступні endpoint-и, структуру запитів і відповідей, а також виконувати тестові звернення до API без запуску Unity-клієнта [1]. Такий підхід зручний саме для backend-частини, оскільки дає змогу окремо перевірити серверну логіку, а вже після цього підключати її до ігрового інтерфейсу.

Основні endpoint-и серверної частини наведено в таблиці 2.1.

Таблиця 2.1 – Основні API-запити серверної частини Spherecage

	Метод і контролер	Призначення та дані
		Створення нового користувача; дані:
		Авторизація наявного користувача; дані: username, password.
		Збереження результату проходження; дані: userId, levelNumber,
		Отримання всіх збережених результатів зі значенням username.
		Отримання загальної таблиці лідерів: 10 найкращих результатів.
		Отримання таблиці лідерів для конкретного рівня.
		Отримання та оновлення прогресу:

Наведений набір запитів покриває мінімальний, але завершений цикл роботи прототипу. Unity-клієнт отримує ідентифікатор користувача після реєстрації або входу, передає його разом із результатом після фінішу рівня, а для відображення рейтингу звертається до leaderboard endpoint-ів.

2.7 Вимоги до backend-частини та обмеження прототипу

Після визначення архітектури, структури даних та основних API-запитів доцільно окремо сформулювати вимоги до backend-частини. Це потрібно для того, щоб реалізація не зводилася лише до набору окремих контролерів, а відповідала конкретним сценаріям гри. Для «Spherecage» вимоги сформовано з урахуванням поточного масштабу прототипу: один рівень, локальний запуск сервера, збереження результату проходження та перегляд таблиці лідерів.

Функціональні вимоги описують, які дії сервер повинен виконувати. До них належать реєстрація або авторизація користувача, збереження результату проходження, отримання leaderboard, підтримка базового прогресу та повернення даних у форматі, який може обробити Unity-клієнт. Нефункціональні вимоги стосуються простоти запуску, зрозумілості структури, стабільності при локальній роботі та можливості подальшого розширення.

Перелік основних вимог наведено в таблиці 2.2.

Таблиця 2.2 – Вимоги до backend-частини прототипу

№	Вимога	Тип і реалізація
	Ідентифікація користувача за введеним іменем	Функціональна; AuthController, User,
	Реєстрація нового користувача у базі даних	Функціональна; POST
	Авторизація існуючого користувача	Функціональна; POST /api/Auth/login.
	Збереження часу проходження рівня	Функціональна; POST /api/Results,
	Отримання таблиці лідерів	Функціональна; GET /api/Leaderboard.
	Підтримка прогресу гравця	Функціональна; ProgressController,
	Локальний запуск без окремого SQL-сервера	Нефункціональна; SQLite-файл
	Зрозуміла структура серверного проєкту	Нефункціональна; Controllers, Models,
	Можливість подальшого розширення API	Нефункціональна; окремі контролери та endpoint за рівнем.

Сформульовані вимоги показують, що backend-частина прототипу має невеликий, але завершений набір обов’язків. Вона не повинна керувати фізикою гри або рухом кулі, оскільки ці задачі залишаються на стороні Unity. Натомість сервер має гарантувати, що дані користувача і результати проходження не

втрачаються після завершення ігрової сесії та можуть бути використані для формування рейтингу.

У поточному стані прототип має і певні обмеження. API працює локально, тому не враховує питання публічного розгортання, захищеного з'єднання для зовнішніх користувачів і масштабування на велику кількість одночасних запитів. Також у маршрутах поки не винесено явну версію API. Для передзахисту та демонстрації ці обмеження є прийнятними, оскільки основна мета полягає у підтвердженні працездатної клієнт-серверної взаємодії, а не у створенні промислового онлайн-сервісу.

Водночас структура реалізації не заважає розвитку системи. За потреби можна додати окремий шар сервісів між контролерами і GameDbContext, ввести DTO для відповідей, розширити перевірку вхідних даних, підключити JWT-авторизацію та винести адресу сервера у конфігураційний файл Unity-клієнта. Такі кроки є логічним продовженням роботи після завершення базового прототипу.

У другому розділі було розглянуто питання, пов'язані з проєктуванням серверної частини гри «Spherecage». Спочатку було визначено загальну архітектуру системи, у межах якої клієнтська частина, реалізована в Unity, взаємодіє із сервером через Web API, а збереження даних виконується в базі SQLite. Такий підхід дозволив відокремити ігрову логіку від серверної логіки та побудувати систему з чітким розподілом відповідальності між її компонентами.

Далі було спроєктовано структуру даних і програмний інтерфейс взаємодії між клієнтом і сервером. Визначено основні сутності системи, а також набір API-запитів, які забезпечують реєстрацію користувача, авторизацію, збереження результатів та отримання таблиці лідерів. Це дозволило сформувати основу для стабільної роботи серверної частини без зайвого ускладнення структури проєкту.

Окремо було обґрунтовано вибір технологій, використаних під час реалізації системи. Використання ASP.NET Core Web API, SQLite та Entity Framework Core виявилося доцільним для навчального клієнт-серверного

проєкту, оскільки такий стек забезпечує достатню функціональність, простоту розгортання та зручність інтеграції з Unity.

Також у розділі було описано структуру серверної частини проєкту та основні сценарії взаємодії клієнта і сервера. Це дозволило показати, як саме окремі компоненти системи працюють разом і яким чином реалізується повний цикл: від введення імені користувача до збереження результату та його відображення у таблиці лідерів.

Отже, у другому розділі було сформовано проєктну основу серверної частини гри «Spherecage», яка надалі використовується для програмної реалізації та перевірки працездатності системи.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ BACKEND-ЧАСТИНИ ГРИ

3.1 Реалізація серверної частини

Практична реалізація backend-частини виконана як окремий ASP.NET Core Web API проєкт. Такий підхід дозволяє відділити ігрову логіку Unity від обробки даних і зробити серверну частину самостійним компонентом системи. У проєкті сервер відповідає за приймання HTTP-запитів, перевірку вхідних даних на базовому рівні, роботу з базою даних SQLite та повернення результатів у форматі

Точкою входу серверної частини є файл Program.cs. У ньому підключаються контролери, Swagger/OpenAPI для перевірки API в режимі розробки, а також контекст бази даних GameDbContext. Для доступу до бази використано рядок підключення Data Source=spherecage.db, тобто дані зберігаються у локальному файлі SQLite. Це відповідає масштабу прототипу: систему можна запустити локально без окремого серверу баз даних, а всі дані користувачів і проходжень залишаються доступними між запусками.

Серверна частина організована за простою структурою: моделі описують сутності бази даних, DTO-класи описують дані, які надходять у запитах, контролери реалізують endpoint-и, а GameDbContext відповідає за зв’язок між C#-класами та таблицями SQLite. Така структура не перевантажує прототип зайвими шарами, але дає зрозумілий розподіл відповідальності між файлами.

У контролері AuthController реалізовано два основні сценарії: реєстрацію нового користувача та вхід існуючого користувача. Під час реєстрації сервер перевіряє наявність імені користувача і пароля, контролює унікальність імені, створює запис у таблиці Users і формує початковий запис прогресу в PlayerProgresses. Під час входу сервер хешує отриманий пароль і порівнює його зі збереженим значенням. У разі успіху клієнт отримує userId, який надалі використовується для збереження результатів.

У межах прототипу пароль у Unity-клієнті задається спрощено, а хешування виконується за допомогою SHA-256. Для прототипної реалізації

цього достатньо, щоб не зберігати пароль у відкритому вигляді. Водночас у промисловій системі такий механізм потрібно було б посилити: використовувати сіль, спеціалізований алгоритм хешування паролів і повноцінну авторизацію через токени. У роботі це розглядається як напрям подальшого розвитку, а не як недолік, що заважає роботі поточного прототипу.

Контролер ResultsController відповідає за збереження результатів проходження рівня. Після фінішу Unity-клієнт передає на сервер ідентифікатор користувача, номер рівня та час проходження. Сервер створює об'єкт GameResult, додає до нього дату завершення у форматі UTC і зберігає запис у базі даних. Додатковий GET-запит у цьому контролері дозволяє отримати всі результати, відсортовані за номером рівня та часом проходження.

Контролер LeaderboardController формує таблицю лідерів. Запит /api/Leaderboard повертає десять найкращих результатів за часом проходження, а запит /api/Leaderboard/{levelNumber} дозволяє отримати рейтинг для конкретного рівня. Для поточної версії гри, де реалізовано один рівень, достатньо загального leaderboard, однак наявність endpoint-а з номером рівня спрощує майбутнє розширення гри.

Контролер ProgressController реалізує зберігання базового прогресу гравця: відкритий рівень, останній завершений рівень і час оновлення. У поточному прототипі цей функціонал не є центральним для проходження одного рівня, але він показує, як backend може підтримувати розвиток гри з декількома рівнями без зміни загальної архітектури.

3.2 Реалізація роботи з базою даних

Для роботи з даними використано Entity Framework Core, який дозволяє описати структуру бази через C#-моделі та виконувати запити до SQLite без ручного написання SQL для кожної операції [2]. У GameDbContext оголошено три набори даних: Users, PlayerProgresses і GameResults. Також у методі OnModelCreating описано унікальний індекс для імені користувача та зв'язки між користувачем, прогресом і результатами.

Модель User містить ідентифікатор, ім’я користувача та хеш пароля. Модель GameResult зберігає ідентифікатор результату, userId, номер рівня, час проходження та дату завершення. Модель PlayerProgress містить інформацію про відкритий рівень і останній завершений рівень. Такий набір сутностей відповідає мінімальним потребам гри: визначити користувача, зафіксувати проходження і повернути рейтинг.

SQLite обрано як базу даних через простоту запуску, локальне зберігання у файлі та достатність для невеликого прототипу [3]. Для дипломного проєкту це практично, оскільки сервер можна перевіряти на одному комп’ютері без налаштування окремого SQL-сервера. При цьому структура з EF Core не прив’язує всю логіку жорстко до SQLite: у разі потреби базу даних можна замінити на іншу, зберігши більшість моделей і контролерів.

Структуру бази даних наведено в таблиці 3.1.

Таблиця 3.1 – Основні таблиці бази даних Spherecage

Сутність	Ключові поля	Призначення та зв’язки
		Зберігання зареєстрованих користувачів; Username має унікальний індекс.
		Збереження результатів проходження рівнів; UserId пов’язаний із Users.Id.
		Збереження базового прогресу гравця; UserId пов’язаний із

Наведена структура є достатньою для поточного прототипу, оскільки вона підтримує повний цикл: створення користувача, проходження рівня, збереження результату та відображення рейтингу. Окреме зберігання прогресу дозволяє розширити гру кількома рівнями без зміни базової моделі користувача.

3.3 Інтеграція Unity-клієнта з Web API

На стороні Unity взаємодія з сервером зосереджена переважно у скрипті ApiClient.cs. У ньому задано базову адресу сервера http://127.0.0.1:5007, формуються JSON-об’єкти запитів і виконуються HTTP-звернення через

UnityWebRequest [4]. Такий підхід дозволяє залишити всю мережеву логіку в одному місці та не дублювати роботу з API у різних ігрових скриптах.

Після запуску гри StartPanelController зупиняє ігровий час через Time.timeScale = 0 і очікує введення імені користувача. Коли гравець натискає кнопку старту, клієнт викликає RegisterOrLogin. Спочатку виконується запит на реєстрацію. Якщо користувач із таким іменем уже існує, сервер повертає помилку, після чого клієнт автоматично виконує запит на вхід. Для користувача це виглядає як один простий сценарій: ввести ім’я і почати гру.

Після успішної реєстрації або авторизації отримані userId та username зберігаються у PlayerSession. Цей об’єкт не знищується під час переходів між сценами, тому ідентифікатор користувача залишається доступним для інших скриптів. Саме userId надалі використовується у FinishZone для відправлення результату проходження.

Коли гравець досягає фінішної зони, FinishZone зупиняє таймер, отримує час проходження, визначає номер рівня і викликає SendGameResult. На сервер передається userId, levelNumber і completionTime. Якщо ApiClient відсутній або userId недійсний, результат не відправляється, а в консоль Unity виводиться повідомлення про помилку. Це дозволяє швидко виявити проблему під час тестування сцени.

Для відображення рейтингу використано LeaderboardUI. Скрипт виконує GET-запит до /api/Leaderboard, отримує JSON-масив результатів, перетворює його на набір об’єктів LeaderboardEntry і формує текстовий список для UI. У разі помилки користувач бачить повідомлення про неможливість завантаження таблиці лідерів.

UML-діаграму послідовності взаємодії клієнта і сервера під час збереження результату наведено на рисунку 3.1.

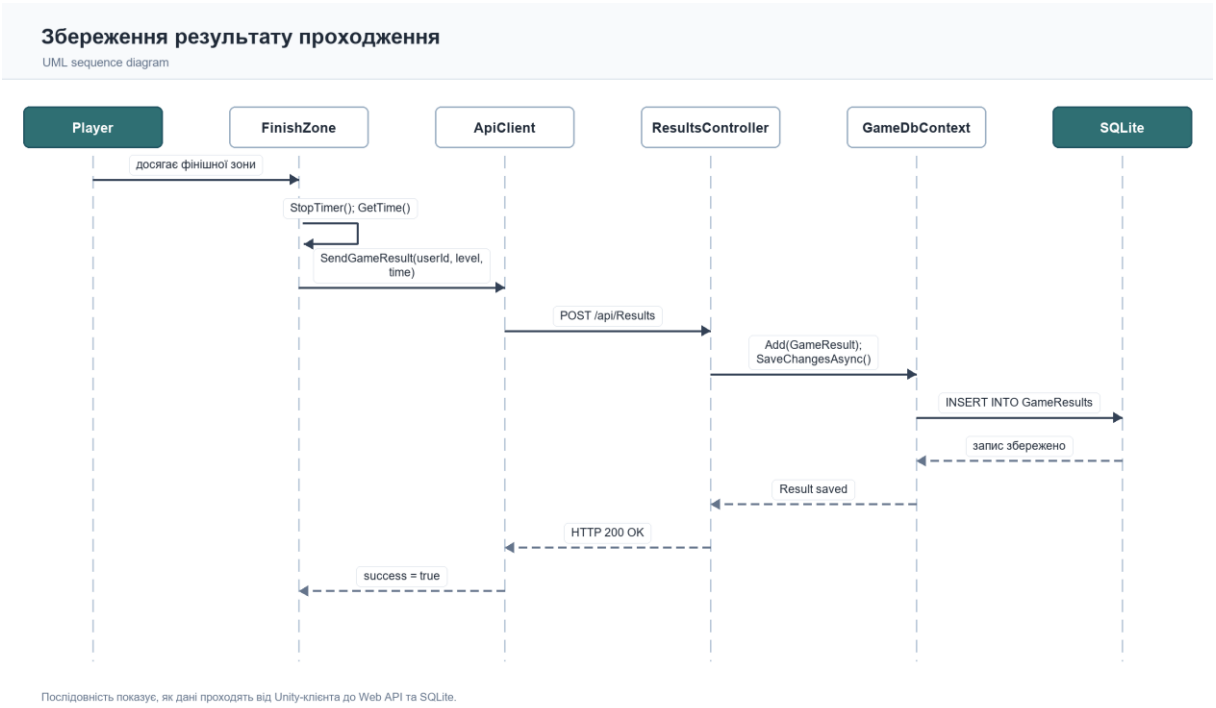

Рисунок 3.1 – UML-діаграма послідовності збереження результату проходження рівня

UML-діаграму послідовності отримання таблиці лідерів наведено на рисунку 3.2. Цей рисунок варто використати також у презентації, оскільки саме на етапі отримання leaderboard добре видно повний шлях даних від бази SQLite до інтерфейсу Unity.

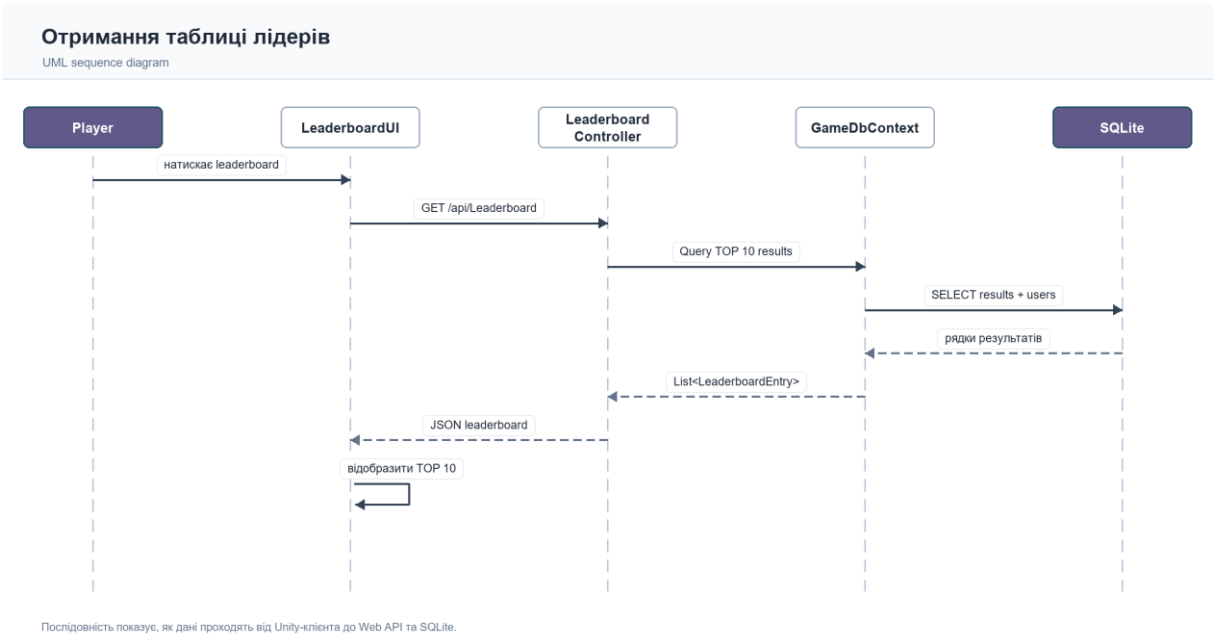

Рисунок 3.2 – UML-діаграма послідовності отримання таблиці лідерів

3.4 Тестування серверної частини

Тестування виконувалося з урахуванням того, що проєкт є прототипом і має обмежений набір функцій. Основна мета тестування полягала не в побудові великої автоматизованої тестової системи, а в перевірці критичних сценаріїв, без яких клієнт-серверна частина гри не може вважатися працездатною.

Перевірка проводилася у кілька етапів. Спочатку тестувалися окремі API-запити через Swagger та HTTP-звернення до локального сервера. Потім перевірялася взаємодія Unity-клієнта з API під час запуску гри, проходження рівня і відкриття таблиці лідерів. Окремо було змодельовано просте навантаження для endpoint-а leaderboard, оскільки саме цей запит може виконуватися частіше під час перегляду результатів різними користувачами.

Результати функціонального тестування наведено в таблиці 3.2.

Таблиця 3.2 – Результати функціонального тестування API

Сценарій	Запит і очікування	Результат
Реєстрація нового користувача	POST /api/Auth/register; очікується створення користувача та повернення	Запит виконано успішно; пройдено.
Повторний вхід	POST /api/Auth/login; очікується повернення даних існуючого користувача.	Авторизація виконана; пройдено.
Збереження результату	POST /api/Results; очікується додавання результату до GameResults.	Запис збережено у SQLite; пройдено.
Отримання всіх результатів	GET /api/Results; очікується список проходжень.	Список отримано та відсортовано; пройдено.
Таблиця лідерів	GET /api/Leaderboard; очікується 10 найкращих результатів.	Leaderboard повернуто у JSON; пройдено.
Leaderboard за рівнем	GET /api/Leaderboard/1; очікуються результати першого рівня.	Фільтрація за рівнем працює; пройдено.
Прогрес гравця	GET/POST /api/Progress; очікується читання та оновлення прогресу.	Запити виконуються для існуючого userId; пройдено.

Функціональна перевірка показала, що основні endpoint-и виконують потрібні дії та повертають дані, які може обробити Unity-клієнт. Найважливішими для поточної версії є сценарії авторизації, збереження результату і отримання таблиці лідерів, оскільки саме вони формують завершений ігровий цикл.

Для додаткової перевірки було змодельовано просте навантаження на запит отримання таблиці лідерів. Тест складався з 50 послідовних GET-запитів до /api/Leaderboard на локальному сервері. Під час тесту не було зафіксовано помилок, середній час відповіді становив 1,4 мс, 95-й процентиль – 2,5 мс, максимальний час відповіді – 10,6 мс. Для невеликого прототипу такі результати підтверджують, що endpoint leaderboard працює стабільно в умовах простого локального навантаження.

Результати модельованого навантаження наведено в таблиці 3.3.

Таблиця 3.3 – Результати модельованого навантажувального тестування

Показник	Значення
Сценарій	50 послідовних GET-запитів до /api/Leaderboard
Кількість помилок	
Середній час відповіді	1,4 мс
95-й процентиль	2,5 мс
Максимальний час відповіді	10,6 мс

Наведені показники не слід трактувати як повноцінне промислове навантажувальне тестування, оскільки перевірка виконувалася локально і без великої кількості паралельних користувачів. Однак для прототипу цього достатньо, щоб підтвердити відсутність очевидних проблем у найчастішому запиті читання даних.

Під час тестування також було визначено кілька обмежень поточної реалізації. По-перше, у ResultsController доцільно додати перевірку існування користувача перед збереженням результату. По-друге, механізм паролів у прототипі є спрощеним і потребує покращення для реального використання. По-третє, явне версіонування API у маршрутах поки не реалізовано, тому його варто додати перед подальшим розширенням серверної частини. Ці обмеження не заважають роботі поточної версії, але є корисними напрямами розвитку.

Для тестування були використані як реальні дії у Unity-клієнті, так і змодельовані дані, які дозволяють перевірити API без багаторазового проходження рівня вручну. Такий підхід є практичним для прототипу: основний

ігровий сценарій перевіряється через клієнт, а додаткові варіанти даних швидко відпрацьовуються на рівні backend.

Тестові дані добиралися так, щоб охопити нормальні сценарії та прості граничні випадки: новий користувач, повторний користувач, результат першого рівня, короткий час проходження, порожня таблиця лідерів і повторне отримання leaderboard після додавання нового результату. Приклади таких даних наведено в таблиці 3.4.

Таблиця 3.4 – Тестові дані для перевірки backend-частини

№	Дані	Використання та очікувана поведінка
		AuthController; створення або вхід користувача.
		AuthController; хешування і порівняння з базою.
		FinishZone, ResultsController; результат дозволено відправити.
		ultsController, LeaderboardController; результат прив’язано до першого рівня.
		GameResultDto; час збережено як числове значення.
	50 запитів GET	LeaderboardController; відповіді без помилок.

Окрему увагу під час тестування було приділено не лише успішним сценаріям, а й реакції системи на помилки. Для невеликого прототипу достатньо перевірити базові негативні випадки: порожні дані авторизації, повторну реєстрацію існуючого користувача, неправильний пароль і спробу отримати прогрес для неіснуючого користувача.

Таблиця 3.5 – Перевірка негативних сценаріїв

Сценарій	Запит і відповідь	Коментар
Порожнє ім’я або пароль		Сервер не створює некоректного користувача.
Повторна реєстрація		Unity після цього переходить до
Неправильний пароль		Користувач не отримує userId.
Відсутній прогрес		Сервер явно повідомляє про відсутність запису.
Недійсний userId у Unity	FinishZone; результат не надсилається.	Клієнт виводить помилку в консоль.

Негативні сценарії важливі для пояснення меж поточної реалізації. Вони показують, що сервер уже має базові перевірки для авторизації та прогресу, але

збереження результатів потребує додаткової валідації userId. Це не порушує роботу поточного Unity-сценарію, оскільки клієнт надсилає результат лише після успішного отримання ідентифікатора користувача, проте для розширеної версії API така перевірка має бути додана на сервері.

3.5 Оцінка результатів реалізації

У результаті реалізації створено backend-частину, яка забезпечує основні потреби гри «Spherecage». Сервер приймає дані від Unity-клієнта, зберігає їх у SQLite, повертає результати у форматі JSON і дозволяє сформувати таблицю лідерів. Завдяки цьому прототип не обмежується локальним проходженням рівня, а має централізоване збереження користувацьких результатів.

Важливо, що backend не втручається безпосередньо у фізику руху кулі та проходження лабіринту. Ця логіка залишається на стороні Unity, де працюють Rigidbody, тригери, таймер і UI. Сервер підключається у ключових точках: коли потрібно визначити користувача, зафіксувати завершення рівня або отримати рейтинг. Такий розподіл відповідальності є зрозумілим і практичним для невеликої гри.

Реалізація також залишає простір для розвитку. У майбутньому можна додати явне версіонування API, розширити таблицю прогресу для більшої кількості рівнів, реалізувати JWT-авторизацію, покращити валідацію запитів і перейти до більш потужної бази даних, якщо кількість користувачів зростатиме. Однак для поточної мети – створення працездатного прототипу гри з backend-складовою – реалізований набір функцій є достатнім.

Для наочного пояснення реалізації доцільно подати не лише текстовий опис, а й UML-діаграми. Компонентна діаграма показує розподіл відповідальності між Unity-клієнтом, контролерами Web API, Entity Framework Core та SQLite, а діаграма класів показує структуру основних сутностей бази даних.

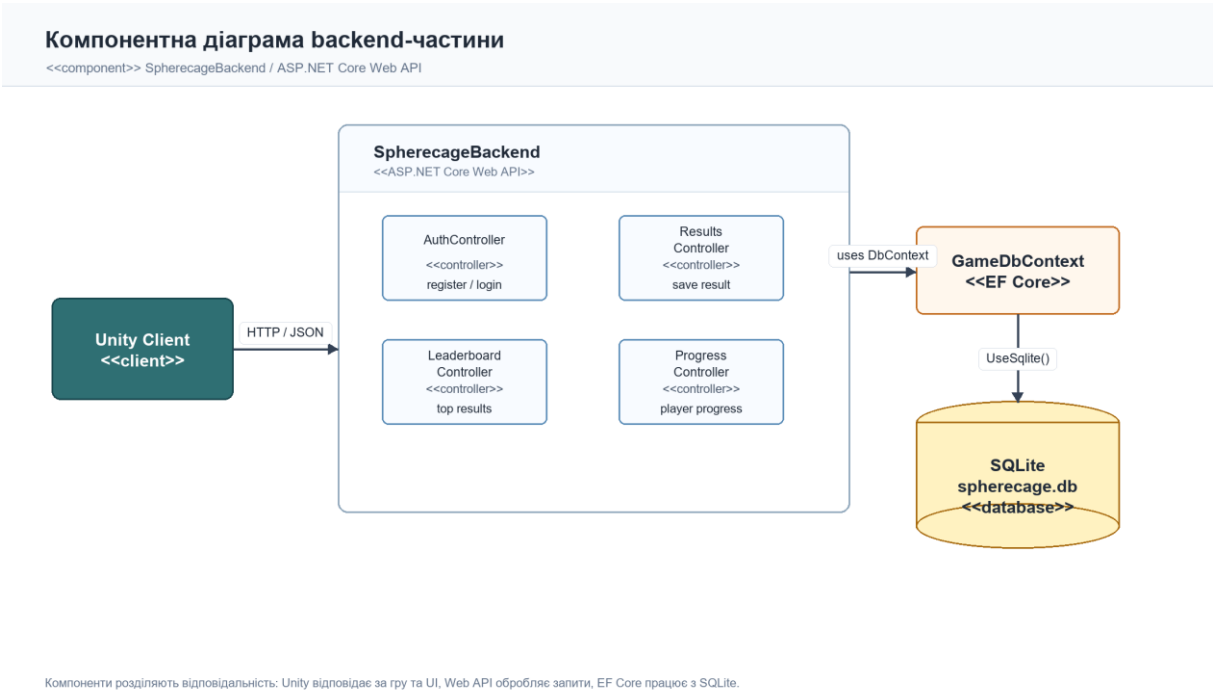

Рисунок 3.3 – UML-діаграма компонентів backend-частини гри «Spherecage»

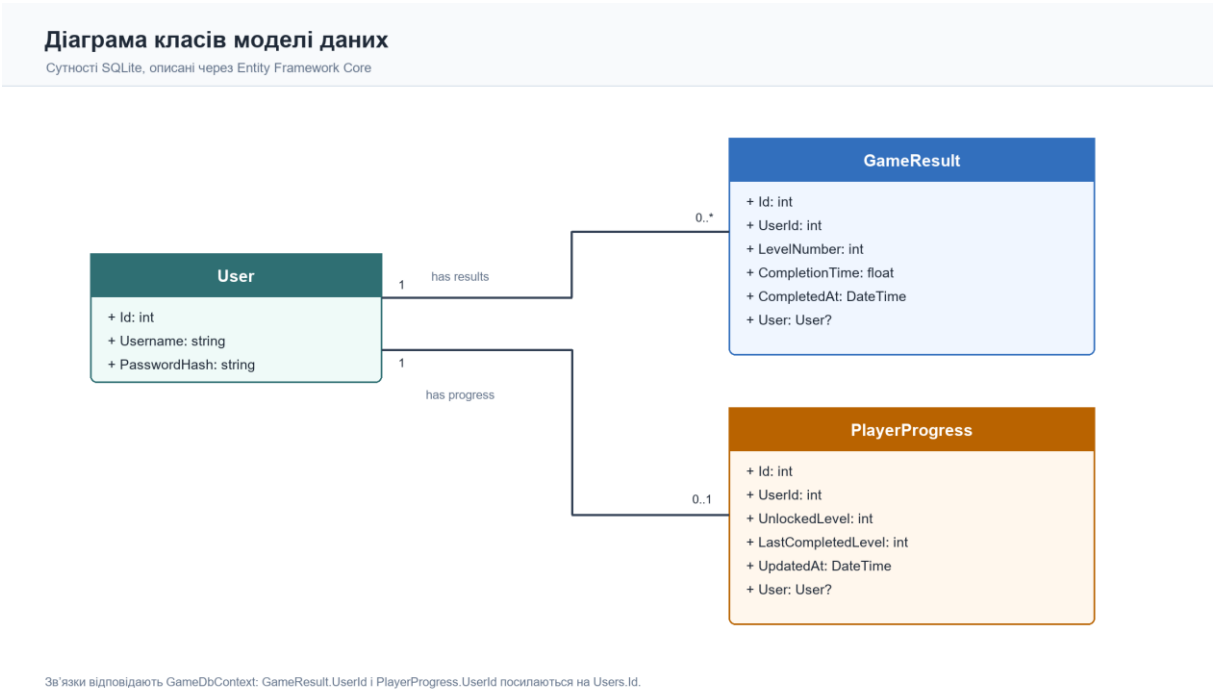

Рисунок 3.4 – UML-діаграма класів моделі даних backend-частини

3.6 Обмеження прототипу та напрями розвитку API

Поточна backend-частина створена як прототип, тому частина рішень свідомо залишена простою. Це дозволило швидко отримати працездатну систему та не ускладнювати дипломний проєкт функціями, які не є критичними для

демонстрації основної ідеї. Однак під час підготовки фінальної версії роботи важливо не приховувати ці обмеження, а чітко показати, які з них є прийнятними для прототипу і що саме можна покращити у майбутньому.

Найважливішим напрямом розвитку є посилення механізму авторизації. Зараз клієнт використовує спрощений пароль, а сервер повертає ідентифікатор користувача без токена доступу. Для локальної демонстрації цього достатньо, але для публічного використання потрібно реалізувати окремий механізм сесій або JWT-токени, зберігати пароль із сіллю та додати обмеження на некоректні запити.

Другим напрямом є валідація результатів. Поточний сервер приймає userId, levelNumber і completionTime від клієнта. У майбутній версії доцільно перевіряти, чи існує користувач, чи відкритий відповідний рівень, чи не є час проходження від'ємним або нереалістично малим. Це дозволить зробити таблицю лідерів більш надійною.

Третій напрям стосується версіонування API. Поточна структура /api/[controller] зручна для невеликого прототипу, але при додаванні нових можливостей краще використовувати явну версію маршруту, наприклад /api/v1/Results. Тоді нові endpoint-и можна буде додавати у v2 без поломки старого Unity-клієнта.

Узагальнення можливих покращень наведено в таблиці 3.6.

Таблиця 3.6 – Напрями подальшого розвитку backend-частини

Напрям	Поточний стан	Покращення та користь
Авторизація	Спрощений пароль і	JWT, сіль для паролів, окремі DTO відповідей; кращий захист користувацьких даних.
Валідація результатів	Результат приймається від клієнта.	Перевірка userId, рівня і допустимого часу; надійніша таблиця лідерів.
Версіонування API	Маршрути	Маршрути /api/v1/...; безпечне додавання нових версій.
Масштабування	Локальний SQLite-файл.	Перехід на серверну СУБД за потреби; підтримка більшої кількості користувачів.
Тестування	Ручні та змодельовані перевірки.	Автоматизовані інтеграційні тести; швидша перевірка після змін.
Unity-конфігурація	Адреса API у скрипті.	Винесення baseUrl у налаштування; простіше розгортання на різних машинах.

Наведені напрями розвитку не змінюють сутність виконаної роботи, а показують, що поточна архітектура має зрозуміле продовження. Для фінальної кваліфікаційної роботи важливо, що базовий backend уже працює, інтегрується з Unity-клієнтом і зберігає дані у SQLite. Подальші зміни можуть виконуватися поступово, без повного переписування проєкту.

У третьому розділі описано практичну реалізацію backend-частини гри «Spherecage». Розглянуто структуру серверного проєкту, роботу контролерів, взаємодію з базою даних SQLite через Entity Framework Core та інтеграцію Unity-клієнта з Web API. Показано, що серверна частина забезпечує реєстрацію й авторизацію користувачів, збереження результатів проходження, отримання таблиці лідерів і базову роботу з прогресом гравця.

Проведене тестування підтвердило працездатність ключових сценаріїв. Основні API-запити виконуються коректно, результати зберігаються у базі даних, а leaderboard повертає відсортований список найкращих проходжень. Модельоване навантаження на endpoint таблиці лідерів не виявило помилок і показало достатню швидкість відповіді для локального прототипу. Водночас визначено напрями подальшого покращення: посилення авторизації, додавання валідації результатів, явне версіонування API та розширення підтримки багаторівневої структури гри.

ВИСНОВКИ

У кваліфікаційній роботі виконано розробку backend-частини прототипу гри-головоломки «Spherecage». У межах роботи було проаналізовано предметну область ігор із фізичною взаємодією, визначено потребу в серверній складовій для збереження користувачів, результатів проходження та формування таблиці лідерів.

Під час проєктування було обрано клієнт-серверну архітектуру, у якій Unity відповідає за ігрову сцену, керування кулею, таймер, фініш і UI, а серверна частина відповідає за обробку запитів, роботу з базою даних і повернення даних клієнту. Такий розподіл дозволив не перевантажувати Unity логікою збереження даних і зробити backend окремим зрозумілим компонентом системи.

Серверну частину реалізовано на основі ASP.NET Core Web API. Для доступу до даних використано Entity Framework Core, а як базу даних обрано SQLite. У структурі бази передбачено таблиці користувачів, результатів проходження та прогресу гравця. Реалізовані контролери AuthController, ResultsController, LeaderboardController і ProgressController забезпечують основні сценарії роботи прототипу.

У роботі реалізовано взаємодію Unity-клієнта з сервером через HTTP-запити у форматі JSON. Після введення імені користувача клієнт виконує реєстрацію або авторизацію, після завершення рівня надсилає результат проходження на сервер, а під час відкриття таблиці лідерів отримує від backend відсортований список результатів. Таким чином сформовано завершений цикл роботи: користувач входить у гру, проходить рівень, результат зберігається і стає доступним у leaderboard.

Проведене тестування підтвердило працездатність основних API-запитів. Було перевірено реєстрацію, авторизацію, збереження результату, отримання всіх результатів, отримання таблиці лідерів і роботу з прогресом. Додатково змодельовано 50 послідовних запитів до endpoint-а /api/Leaderboard, під час яких не було зафіксовано помилок, а середній час відповіді становив 1,4 мс.

Практичне значення отриманих результатів полягає у створенні працездатної backend-основи для розважального ігрового контенту. Реалізована система може бути використана для подальшого розвитку гри «Spherecage», додавання нових рівнів, розширення таблиці лідерів, покращення авторизації та розвитку унікальної механіки гри, натхненної ідеєю керування кулею у просторовому лабіринті.

Отже, поставлену мету роботи досягнуто: серверна частина прототипу гри «Spherecage» спроєктована, реалізована, інтегрована з Unity-клієнтом і перевірена на основних сценаріях використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Microsoft. ASP.NET Core documentation : веб-сайт. URL: https://learn.microsoft.com/en-us/aspnet/core/ (дата звернення: 23.06.2026).
2. Microsoft. Create web APIs with ASP.NET Core : веб-сайт. URL: https://learn.microsoft.com/en-us/aspnet/core/web-api/ (дата звернення: 23.06.2026).
3. Microsoft. Tutorial: Create a controller-based web API with ASP.NET Core : веб-сайт. URL: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api (дата звернення: 23.06.2026).
4. Microsoft. Routing to controller actions in ASP.NET Core : веб-сайт. URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/routing (дата звернення: 23.06.2026).
5. Microsoft. Controller action return types in ASP.NET Core web API : веб-сайт. URL: https://learn.microsoft.com/en-us/aspnet/core/web-api/action-return-types (дата звернення: 23.06.2026).
6. Microsoft. ASP.NET Core web API documentation with Swagger / OpenAPI : веб-сайт. URL: https://learn.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger (дата звернення: 23.06.2026).
7. NuGet. Swashbuckle.AspNetCore package : веб-сайт. URL: https://www.nuget.org/packages/Swashbuckle.AspNetCore (дата звернення: 23.06.2026).
8. Microsoft. Entity Framework Core documentation : веб-сайт. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 23.06.2026).
9. Microsoft. SQLite EF Core Database Provider : веб-сайт. URL: https://learn.microsoft.com/en-us/ef/core/providers/sqlite/ (дата звернення: 23.06.2026).
10. Microsoft. Migrations Overview - EF Core : веб-сайт. URL: https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/ (дата звернення: 23.06.2026).

11. SQLite. SQLite Documentation : веб-сайт. URL: https://sqlite.org/docs.html (дата звернення: 23.06.2026).

12. SQLite. SQL As Understood By SQLite : веб-сайт. URL: https://sqlite.org/lang.html (дата звернення: 23.06.2026).

13. Microsoft. Serialize and deserialize JSON using C# : веб-сайт. URL: https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/overview (дата звернення: 23.06.2026).

14. Microsoft. C# language documentation : веб-сайт. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 23.06.2026).

15. Unity Technologies. Unity Documentation : веб-сайт. URL: https://docs.unity.com/ (дата звернення: 23.06.2026).

16. Unity Technologies. Unity User Manual : веб-сайт. URL: https://docs.unity3d.com/6000.4/Documentation/Manual/UnityManual.html (дата звернення: 23.06.2026).

17. Unity Technologies. Unity Manual: UnityWebRequest : веб-сайт. URL: https://docs.unity3d.com/Manual/UnityWebRequest.html (дата звернення: 23.06.2026).

18. Unity Technologies. Rigidbody component reference : веб-сайт. URL: https://docs.unity3d.com/6000.4/Documentation/Manual/class-Rigidbody.html (дата звернення: 23.06.2026).

19. Unity Technologies. Input System : веб-сайт. URL: https://docs.unity3d.com/6000.4/Documentation/Manual/com.unity.inputsystem.html (дата звернення: 23.06.2026).

20. MDN Web Docs. HTTP: Hypertext Transfer Protocol : веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP (дата звернення: 23.06.2026).

21. MDN Web Docs. HTTP request methods : веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods (дата звернення: 23.06.2026).

22. Valve Corporation. Ballance on Steam : веб-сайт. URL: https://store.steampowered.com/app/2000770/Ballance/ (дата звернення: 23.06.2026).

23. SEGA. Super Monkey Ball Banana Mania Official Site : веб-сайт. URL: https://smb.sega.com/bananamania/ (дата звернення: 23.06.2026).

24. MobyGames. Marble Madness (1984) : веб-сайт. URL: https://www.mobygames.com/game/466/marble-madness/ (дата звернення: 23.06.2026).

ДОДАТОК А

Посилання на репозиторій програмного проєкту

Репозиторій клієнт-серверної 3D гри-головоломки «Spherecage» розміщено у відкритому доступі на платформі GitHub.

У репозиторії наведено вихідний код Unity-клієнта, серверної частини ASP.NET Core Web API, файли налаштувань проєкту та супровідні матеріали, необхідні для ознайомлення з реалізацією проєкту.

Посилання: https://github.com/komradi/Spherecage