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

КВАЛІФІКАЦІЙНА РОБОТА

на тему

РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ 2D ГРИ ТАНКОВИЙ БІЙ

Виконав: студент групи 1К-21
Спеціальності 123 Комп’ютерна інженерія
Максим УСАНІН
Керівник:
Дмитро ПОДОРОШКО

Черкаси 2025

ЗМІСТ

ВСТУП

У XXI столітті цифрові технології стрімко інтегруються в усі сфери життя людини, зокрема в індустрію відеоігор. Ігрова індустрія є однією з най динамічніших галузей програмного забезпечення, яка щорічно залучає мільйони користувачів по всьому світу. Особливе місце в цій галузі займають багатокористувацькі онлайн-ігри, які потребують ефективної побудови архітектури клієнт-сервер, де критично важливою є серверна частина.

Серверна частина гри виконує низку важливих функцій, зокрема керування гравцями, обробку подій, синхронізацію стану гри між клієнтами, збереження результатів і статистики. Розробка такої компоненти вимагає застосування сучасних технологій програмування та відповідних протоколів для ефективної мережевої взаємодії. У цьому контексті особливої актуальності набувають Web API, які забезпечують обмін даними між клієнтом і сервером у режимі реального часу.

Актуальність теми полягає в необхідності розробки якісного серверного забезпечення для двовимірної гри з мережевим режимом, що здатне обробляти множинні підключення користувачів та забезпечувати стабільність функціонування в реальних умовах. Існуючі приклади таких систем, зокрема класичні ігри на кшталт «Battle City» для прикладу клієнтської частини чи сучасні інді-ігри, демонструють потребу в оптимізованих та безпечних рішеннях.

Мета роботи – розробити серверну частину 2D гри «Танковий бій» з використанням сучасних інструментів (FastAPI, SQLite), що забезпечить ефективну обробку дій гравців, підтримку авторизації користувачів, синхронізацію об'єктів гри та збереження даних у базі.

Об’єкт дослідження – серверна частина веб-орієнтованого багатокористувацького застосунку.

Предмет дослідження – технології та методи розробки ігрових Web API, обробки запитів і синхронізації клієнтів у режимі реального часу.

Методи дослідження – аналіз архітектури клієнт-сервер, проектування REST API, використання реляційної бази даних (SQLite), моделювання логіки гри, тестування навантаження.

Практичне значення полягає у можливості використання створеної серверної частини як основи для розробки повноцінного ігрового продукту, або як навчального прикладу для студентів IT-спеціальностей.

РОЗДІЛ 1
АНАЛІЗ ПРИДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд серверної архітектури для 2D ігор

Серверна архітектура є критичним елементом у створенні багатокористувацьких 2D ігор, оскільки саме вона забезпечує злагоджену взаємодію між усіма підключеними клієнтами, обробляє основну ігрову логіку, зберігає дані користувачів і гарантує цілісність ігрового процесу. У випадку 2D-ігор, які працюють у браузері або на легких клієнтах, серверна частина не тільки відіграє роль контролера стану, а й виконує значну частину логіки, зменшуючи навантаження на клієнт.

Сервер для таких ігор зазвичай реалізується у вигляді веб-сервісу, що надає набір REST API або WebSocket-інтерфейсів для реального часу. Вибір технології залежить від типу гри: покрокова або динамічна. У випадку динамічних аркад, де взаємодія між гравцями відбувається в реальному часі (напр., танковий бій), важлива низька затримка обміну даними, що вимагає оптимізованого та легкого серверного рішення.

Класична серверна архітектура для 2D ігор включає такі компоненти:

— Сервер логіки гри – обробляє всі дії гравців: переміщення, атаки, колізії, тощо.

— Сервер автентифікації – відповідає за реєстрацію, логін, перевірку користувачів.

— База даних – зберігає дані користувачів, ігровий прогрес, рейтинги тощо.

— API-шлюз – приймає запити від клієнта та перенаправляє їх до відповідних сервісів.

— Модуль синхронізації – забезпечує однаковий стан гри на всіх підключених клієнтах (при мультиплеєрі).

Серверна частина може бути реалізована на різних мовах програмування: Node.js, Go, Python, Java. У даній роботі обрано Python через його простоту,

наявність потужних бібліотек і зручну інтеграцію з FastAPI — фреймворком, що дозволяє швидко створювати високопродуктивні веб-сервіси.

Загалом, якісна серверна архітектура забезпечує:

- Стабільну роботу з великою кількістю гравців;
- Захист від шахрайства (античит, перевірка коректності дій);
- Мінімізацію затримок;
- Масштабованість – можливість розширення гри без переписування ядра.

Таким чином, правильне проектування серверної архітектури є основою для надійної ігрової платформи, особливо у випадку багатокористувацьких 2D проектів.

1.2 Аналіз вимог до серверної частини гри

Розробка серверної частини 2D гри «Танковий бій» потребує чіткого формулювання функціональних та нефункціональних вимог до системи. Сервер повинен забезпечити стабільну, надійну та масштабовану взаємодію між гравцем і клієнтською частиною гри, а також гарантувати збереження ігрових даних і коректну обробку подій у режимі реального часу.

Функціональні вимоги:

1. Реєстрація та авторизація користувачів. Сервер повинен надавати ендпоїнти для створення нового облікового запису, входу користувача, перевірки облікових даних та верифікації токенів (якщо використовується JWT або сесії).

2. Збереження та отримання профілю користувача. Необхідно реалізувати API для зчитування та оновлення інформації про гравця: кількість пройдених рівнів, статистику боїв, ім’я, рейтинг тощо.

3. Передача даних про стан гри. Сервер повинен обробляти запити щодо поточного стану рівнів, надання інформації про доступні рівні, збереження результатів проходження.

4. Обробка повідомлень від клієнта. Передбачено взаємодію з подіями, які надсилає клієнт (наприклад, початок гри, завершення рівня, збереження прогресу), з відповідним збереженням у базі даних.

5. Логування подій та дій гравця. З метою де багу та захисту від зловживань необхідно зберігати ключові події: спроби входу, аномальні завершення гри, злом API-запитів.

Нефункціональні вимоги:

1. Продуктивність. Сервер повинен обробляти запити з мінімальною затримкою, особливо у випадку одночасних підключень користувачів. Для цього обрано асинхронний веб-сервер Uvicorn.

2. Масштабованість. Архітектура має бути розширюваною, з можливістю інтегрування додаткових функцій без зміни ядра системи.

3. Захист даних. Необхідно реалізувати базову перевірку введених даних (валидація через Pydantic), захист від SQL-ін'єкцій, а також шифрування паролів користувачів (наприклад, через bcrypt).

4. Стабільність і безвідмовність. Сервер повинен працювати без збоїв при тривалому навантаженні, із можливістю відновлення після збоїв (автоматичний рестарт процесу або логування падінь).

5. Простота розгортання. Сервер має бути здатним до розгортання як у локальному середовищі, так і в хмарі (наприклад, Render), без необхідності складної конфігурації.

Вимоги до структури API:

- Всі запити повинні мати уніфікований формат (JSON).
- Обробка помилок повинна бути централізованою (через middleware або глобальний обробник).
- Підтримка коду стану HTTP (200, 400, 403, 500 тощо) для зворотного зв'язку клієнту.

Таким чином, ефективна серверна частина має відповідати як функціональним потребам гри, так і забезпечувати високі показники

продуктивності, захисту та надійності, що є критично важливими для багатокористувацьких онлайн-додатків.

1.3 Постановка задачі

На основі аналізу предметної області та визначених вимог до серверної частини гри, у рамках даної класифікаційної роботи ставиться задача створити повнофункціональний серверний модуль для 2D гри «Танковий бій» з архітектурою типу клієнт-сервер. Розроблений сервер повинен забезпечити надійне управління користувачами, обробку запитів клієнтів, взаємодію з базою даних, а також синхронізацію ігрового процесу з мінімальною затримкою.

Задача конкретизується у вигляді таких під задач:

1. Реалізувати REST API для взаємодії з клієнтом, що включає:
   - авторизацію та реєстрацію користувача;
   - надання доступу до ігрової інформації;
   - обробку подій гри (проходження рівнів, збереження результатів);
   - оновлення інформації про профіль гравця.
2. Побудувати структуру бази даних (SQLite) для зберігання:
   - облікових записів користувачів (логін, хешований пароль);
   - індивідуальних параметрів (кількість пройдених рівнів, рейтинг, статус);
   - додаткової інформації (наприклад, дата реєстрації, остання активність).
3. Забезпечити валідацію даних за допомогою бібліотеки Pydantic:
   - перевірка формату запитів (JSON-схеми);
   - контроль типів і допустимих значень на вході.
4. Використати FastAPI як основний фреймворк для:
   - обробки асинхронних HTTP-запитів;
   - організації маршрутів;
   - автоматичної генерації документації API (Swagger UI/OpenAPI).

5. Налаштувати середовище запуску з Uvicorn, що забезпечить:

– високу продуктивність;

– підтримку асинхронності;

– можливість розгортання в хмарі або локально.

6. Реалізувати механізм обробки помилок і повідомлень, з підтримкою:

– HTTP-кодів відповіді;

– централізованого логування;

– повернення зрозумілих повідомлень клієнту.

7. Провести тестування основних функцій сервера, зокрема:

– успішну реєстрацію та вхід;

– обробку неправильно сформованих запитів;

– доступ до ігрової інформації відповідно до авторизації.

Таким чином, основна мета – побудувати надійний серверний API-сервіс для підтримки клієнтської частини гри, з можливістю масштабування, розширення та подальшого використання в мережевій багатокористувацькій грі.

РОЗДІЛ 2
ВИБІР СЕРЕДОВИЩА ТА ТЕХНОЛОГІЙ РОЗРОБКИ СЕРВЕРА

2.1 Обґрунтування вибору Python та FastAPI

Для реалізації серверної частини гри «Танковий бій» було обрано мову програмування Python у поєднанні з фреймворком FastAPI. Цей вибір зумовлений рядом технічних, практичних та інфраструктурних переваг, які безпосередньо впливають на швидкість розробки, зручність масштабування, а також ефективність обробки запитів у мережевому середовищі.

Переваги мови Python:

1. Простота синтаксису та читабельність коду. Python має лаконічну та зрозумілу структуру, що полегшує підтримку й масштабування проекту, а також пришвидшує навчання для нових розробників.

2. Широка екосистема бібліотек. Мова має тисячі бібліотек для обробки HTTP-запитів, роботи з базами даних, шифрування, обробки JSON, а також для тестування.

3. Підтримка асинхронності. Завдяки вбудованим механізмам async/await, Python дозволяє реалізувати асинхронну обробку запитів, що особливо важливо для ігрового сервера з великою кількістю одночасних підключень.

4. Кросплатформеність. Python запускається на будь-якій ОС, що дозволяє легко переносити сервер між локальними та хмарними середовищами.

Обґрунтування вибору FastAPI:

FastAPI – це сучасний, високопродуктивний фреймворк для створення REST API на базі Python 3.7+. Його переваги включають:

1. Підтримка асинхронного виконання. FastAPI побудовано на основі Starlette і Pydantic, що забезпечує нативну асинхронність і швидку обробку запитів, не поступаючись продуктивністю Node.js.

2. Автоматична генерація документації. Після запуску сервер автоматично надає документацію до всіх доступних ендпоїнтів у форматі OpenAPI/Swagger, що полегшує відлагодження і тестування.

3. Вбудована валідація даних. Використовуючи Pydantic, FastAPI автоматично перевіряє структуру вхідних запитів, повідомляє про помилки та захищає сервер від некоректних даних.

4. Швидкий старт і зрозумілий інтерфейс. FastAPI дозволяє створити повноцінний REST API за мінімальний час із чистим, зрозумілим синтаксисом.

5. Гнучка інтеграція з базами даних та фронтендом. FastAPI легко поєднується з SQLite, PostgreSQL, Redis, а також з будь-якими фронтенд-застосунками (через CORS та REST-запити).

Практичне застосування в грі:

У грі «Танковий бій» сервер на FastAPI:

обробляє запити авторизації;

- відповідає за отримання/збереження прогресу користувача;
- підтримує доступ до профілю;
- дозволяє легко масштабувати API для майбутньої підтримки мультиплеєру.

Завдяки використанню Python + FastAPI серверна частина гри поєднує швидкість розробки, зручність тестування та ефективність роботи у продакшн-середовищі.

2.2 Роль Uvicorn, Pydantic і SQLite у проєкті

У проєкті серверної частини гри «Танковий бій» було використано три ключові технології: Uvicorn як сервер виконання, Pydantic як засіб валідації даних, і SQLite як база даних. Їх поєднання дозволило побудувати легкий, асинхронний і стабільний сервер, що відповідає вимогам до реального використання у веборієнтованих ігрових застосунках.

Uvicorn – це високопродуктивний сервер для запуску асинхронних Python-застосунків, що підтримує стандарт ASGI (Asynchronous Server Gateway Interface). На відміну від класичних WSGI-серверів, Uvicorn дозволяє повноцінно використовувати асинхронні функції async def, що критично для обробки великої кількості одночасних запитів.

У рамках цього проєкту Uvicorn:

– запускає FastAPI-додаток на локальному хості або в хмарному середовищі;

– обробляє HTTP-запити від клієнтів з мінімальною затримкою;

– підтримує автоматичну перезагрузку серверу під час розробки (режим --reload);

– дозволяє розгортати сервер через CLI: uvicorn main:app --reload.

Завдяки Uvicorn проєкт зберігає простоту налаштування та високу продуктивність.

Pydantic – це бібліотека Python для декларативної перевірки та серіалізації даних на основі типів. Вона є фундаментальною частиною FastAPI, оскільки забезпечує:

– Перевірку вхідних даних. Всі запити від клієнта проходять через Pydantic-схеми, які перевіряють типи, обов’язковість, допустимі значення.

– Генерацію документації. Pydantic-моделі автоматично додаються до OpenAPI-специфікації.

– Серйозну безпеку. Захищає сервер від SQL-ін’єкцій і пошкоджених JSON-об’єктів.

Наприклад, модель UserCreate(BaseModel) може містити поля username: str, password: str, і FastAPI автоматично відкине запити, які не відповідають цим правилам.

Таким чином, Pydantic гарантує цілісність вхідних даних і значно спрощує обробку запитів.

SQLite – це вбудована реляційна база даних, яка не потребує окремого сервера. Її особливості ідеально підходять для проєктів середнього масштабу та демонстраційних ігор.

У грі «Танковий бій» SQLite використовується для:

- зберігання даних користувачів (логін, пароль, прогрес);
- фіксації проходження рівнів;
- обробки профілів і статистики.

Основні переваги SQLite:

- зручність – база являє собою один .db файл;
- відсутність потреби у налаштуванні СУБД;
- сумісність з FastAPI через SQL-бібліотеки (sqlite3, SQLAlchemy, або raw запити).

У нашому проекті використовується файл users.db, що містить таблицю з логінами, хешами паролів та ігровою інформацією.

Завдяки використанню Uvicorn, Pydantic і SQLite серверна частина гри поєднує в собі:

- асинхронну обробку (Uvicorn),
- захист від помилкових запитів (Pydantic),
- просте зберігання ігрових даних (SQLite), що створює ефективну, безпечну та гнучку серверну інфраструктуру.

2.3 API-архітектура та взаємодія з клієнтом

Серверна частина гри «Танковий бій» побудована за принципами REST-архітектури. Це означає, що кожна функція, яку сервер виконує для клієнта, представлена у вигляді HTTP-ендпоїнта, що приймає і повертає дані у форматі JSON. Такий підхід забезпечує просту, зрозумілу та масштабовану структуру взаємодії.

Основні принципи REST API в проекті:

1. Ресурсність.

Усі елементи гри, з якими працює сервер (користувачі, прогрес, рівні), представляються як ресурси з власними унікальними URI-адресами. Наприклад:

– POST /register – створення нового користувача;
– POST /login – авторизація;
– GET /profile/{username} – отримання профілю;
– POST /level/complete – надсилання інформації про завершення рівня.

2. HTTP-методи.

Сервер використовує стандартні методи HTTP:

– GET – отримання даних (профіль, інформація про рівні);
– POST – надсилання даних (реєстрація, збереження прогресу);
– PUT/PATCH – оновлення (за потреби);
– DELETE – видалення (наприклад, облікового запису).

3. Безстанова взаємодія.

Кожен запит має бути самодостатнім, тобто сервер не зберігає інформацію про попередні запити. Це забезпечує масштабованість і дозволяє легко відслідковувати поведінку клієнта.

4. Стандартизовані коди відповіді.

Сервер повертає відповідні HTTP-статуси, наприклад:

– 200 OK – успішна операція;
– 201 Created – створення ресурсу;
– 400 Bad Request – помилка клієнта;
– 401 Unauthorized — невірна авторизація;
– 500 Internal Server Error – внутрішня помилка сервера.

API-взаємодія з клієнтом

Клієнтська частина гри (розробляється окремо) взаємодіє з API через AJAX-запити з фронтенду (JavaScript). Комунікація проходить наступним чином:

1. Реєстрація:

– Клієнт надсилає POST запит на /register з JSON-тілом { "username": "Player1", "password": "12345" }.

– Сервер перевіряє дані, створює обліковий запис, хешує пароль і повертає повідомлення про успіх.

2. Авторизація:

– Запит на /login з тими ж полями.

– У разі успіху – сервер повертає дані профілю (або токен, якщо реалізовано).

3. Робота з рівнями:

– Клієнт надсилає GET /levels – отримує список доступних рівнів.

– Після проходження рівня – POST /level/complete з даними про результат (рівень, статус, час, тощо).

4. Профіль гравця

Запит GET /profile/{username} повертає поточну інформацію про гравця: кількість життів, пройдені рівні, рейтинг.

Структура API в коді

У файлі main.py визначено маршрути FastAPI лістинг 2.3.

Лістинг 2.3 – маршрути FastAPI

```
@app.post("/register")
async def register(user: UserCreate):     …

@app.post("/login")
async def login(user: UserLogin):         …

@app.get("/profile/{username}")
async def get_profile(username: str):     …
```

Це дозволяє клієнтській частині легко взаємодіяти з сервером без потреби глибокого розуміння серверної логіки.

Переваги архітектури:

- Простота інтеграції з будь-яким фронтендом;
- Гнучкість – легко додавати нові ендпоїнти;
- Прозорість взаємодії через автоматичну документацію (Swagger UI);
- Масштабованість – API може бути використаний у мобільній версії гри чи в іншій платформі.

2.4 Структура серверного проєкту 2D_TANKS_SERVER

Проєкт 2D_TANKS_SERVER реалізовано з використанням фреймворку FastAPI, асинхронного сервера Uvicorn та бази даних SQLite. Структура проєкту побудована таким чином, щоб забезпечити логічне розділення клієнтської та серверної частин, спрощену підтримку, а також можливість подальшого масштабування.

Нижче наведено опис основних папок і файлів серверної частини рис.2.1.

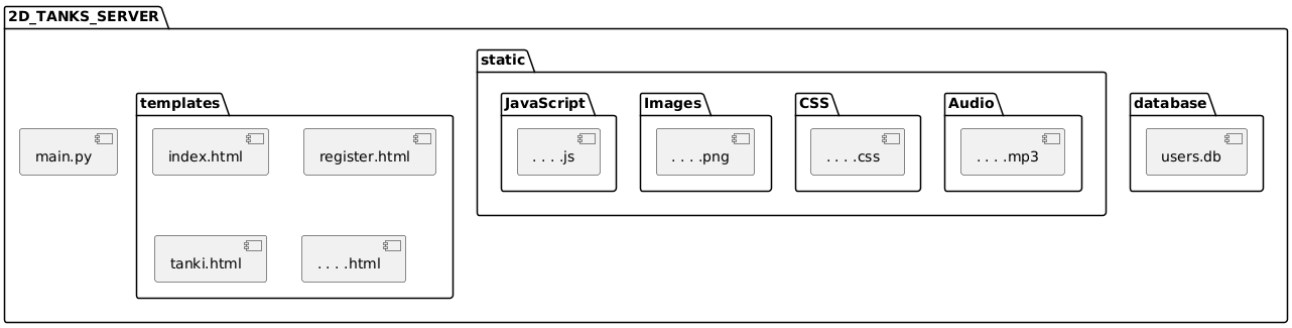

Рисунок 2.1 – Основні папки та файли

Опис основних компонентів main.py

Основний серверний файл, у якому створюється об’єкт FastAPI, визначаються маршрути (@app.post, @app.get), налаштовується робота з HTML-шаблонами (Jinja2Templates) та статичними файлами.

Відповідає за:

- запуск сервера;
- обробку HTTP-запитів (реєстрація, логін, доступ до сторінок);
- рендеринг HTML-сторінок із шаблонів;
- обробку даних з клієнта.

– database/users.db

База даних SQLite, яка містить таблиці користувачів. Містить такі поля, як логін, пароль (у хешованому вигляді), статус, рівень, статистика.

Використовується модуль sqlite3 для роботи з цією базою.

– static/ Каталог для всіх статичних ресурсів (доступних через FastAPI StaticFiles):

– Audio/ – музичні та звукові файли (.mp3);

– CSS/ – таблиці стилів;

– Images/ – графіка, включаючи танки, фон, логотип;

– JavaScript/ – скрипти для обробки взаємодії на клієнтській стороні.

– templates/

Каталог із HTML-файлами, що використовуються для рендерингу сторінок у відповідь на запити клієнта:

– index.html – форма входу;

– register.html – форма реєстрації;

– tanki.html, load_to_game_1.html, load_to_game_2.html – ігрові сторінки.

Взаємодія між компонентами:

1. Користувач надсилає HTTP-запит (наприклад, POST /login).
2. FastAPI (main.py) обробляє запит, перевіряє дані, взаємодіє з users.db.
3. Якщо потрібно – генерується HTML-відповідь через templates/.
4. Статичні ресурси (зображення, CSS, звук) підтягуються з static/.

Переваги обраної структури:

– Чітке розділення логіки: сервер (main.py), база даних, ресурси та шаблони зберігаються окремо.

– Масштабованість: можна додати нові ендпоїнти або базу даних без зміни основної логіки.

– Зручність обслуговування: кожен компонент ізольовано, що полегшує тестування та налагодження.

У підсумку, така структура проекту забезпечує просту, але гнучку архітектуру для невеликої багатокористувацької гри з повноцінною серверною логікою.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ ГРИ

3.1 Реєстрація, авторизація та автентифікація

Реєстрація, авторизація та автентифікація користувачів є ключовими процесами для будь-якої веборієнтованої гри, що зберігає персональний прогрес, статистику або має багатокористувацький режим. У серверній частині гри «Танковий бій» ці механізми реалізовані через HTTP-запити до FastAPI-сервера з використанням бази даних SQLite для зберігання облікових записів.

Реєстрація користувача

Реєстрація – це створення нового облікового запису. Користувач надсилає POST-запит на маршрут /register, у якому передає email, name і password у форматі JSON.

Сервер виконує такі дії:

- перевіряє, чи логін уже існує в базі;
- валідує дані через Pydantic;
- хешує пароль (через bcrypt);
- генерує унікальний п’яти значний id гравця (у hexadecimal системі)
- зберігає новий запис до таблиці users у базі users.db;
- повертає JSON-відповідь із підтвердженням реєстрації.

Цей процес забезпечує унікальність логінів і захист паролів на стороні сервера.

Авторизація користувача

Авторизація – це перевірка даних користувача при вході. Запит відправляється на маршрут /login, також методом POST, і містить логін та пароль.

Сервер:

- шукає відповідний логін у базі;
- перевіряє відповідність введеного пароля з хешем у базі;

– у разі успіху – формує сесію або (опціонально) повертає токен доступу;

– у разі помилки – повертає статус 401 Unauthorized.

У реалізації серверної частини гри токенізація наразі не використовується, автентифікація реалізована через просту перевірку та сесію (або перенаправлення на сторінку гри після входу).

Автентифікація. Автентифікація – це підтвердження того, що користувач є тим, за кого себе видає. У проєкті її забезпечує:

– зберігання пароля в базі лише у хешованому вигляді;

– унікальність id гравця;

– перевірка відповідності введеного пароля до хешу;

– захист маршруту до tanki.html, load_to_game_1.html тощо – тільки для авторизованих користувачів.

У майбутньому можлива інтеграція JWT (JSON Web Token) або OAuth2 для підвищення безпеки, зокрема в мультиплеєрі.

Лістинги(3.1.1 - 3.1.2) фрагментів реалізації в main.py (спрощено):

Лістинг 3.1.1 – Фрагмент коду register

```
@app.post("/register")
async def register(email: str = Form(...), name: str = Form(...), password: str = Form(...)):
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute(
        "INSERT INTO users (user_id, name, email, password) VALUES (?, ?, ?, ?)",
        (generate_user_id(), name, email, hash_value(password))
    )
    conn.commit()
    conn.close()
    return {"message": "Реєстрація успішна"}
```

Лістинг 3.1.2 – Фрагмент коду login

```
@app.post("/login")
async def login(name: str = Form(...), password: str = Form(...)):
    conn = get_db_connection()
    user = conn.execute(
        "SELECT * FROM users WHERE name = ?", (name,)
    ).fetchone()
    conn.close()
    if user and verify_value(password, user["password"]):
        return {"message": " Login successful", "name": user["name"]}
    return {"message": " Invalid login or password"}
```

База даних

У таблиці користувачів зберігаються такі поля:

– user_id (TEXT PRIMARY KEY);
– name (TEXT NOT NULL UNIQUE);
– email (TEXT NOT NULL) ;
– password (TEXT NOT NULL);
– recovery_code (TEXT) .

У таблиці прогресу користувачів зберігаються такі поля:

– user_id (TEXT PRIMARY KEY);
– level (INTEGER NOT NULL);
– score (INTEGER NOT NULL) ;
– stars (INTEGER NOT NULL).

Висновок

Завдяки реалізованим механізмам реєстрації та авторизації, сервер надійно зберігає облікові дані гравців, захищає доступ до ігрових сторінок, а також дозволяє підтримувати персоналізований ігровий досвід. Цей функціонал є основою для реалізації прогресу, рейтингів і багатокористувацької взаємодії.

3.2 Робота з базою даних користувачів (SQLite)

У серверній частині гри «Танковий бій» для зберігання інформації про гравців використовується реляційна база даних SQLite. Цей формат обрано як оптимальний варіант для локальних і мало масштабних веб застосунків, оскільки SQLite:

– не потребує окремого сервера або додаткових сервісів;

– має малий обсяг і зберігається у вигляді одного .db файлу;

– підтримується стандартними бібліотеками Python (sqlite3);

– забезпечує достатню продуктивність для зберігання облікових записів та прогресу.

Структура таблиці користувачів

У файлі users.db зберігається таблиця users, з наступною структурою. Код створення таблиці users лістинг 3.2.

Лістинг 3.2 – Створення таблиці users

```
# users
db.execute("""
  CREATE TABLE IF NOT EXISTS users (
    user_id TEXT PRIMARY KEY,
    name TEXT NOT NULL UNIQUE,
    email TEXT NOT NULL,
    password TEXT NOT NULL,
    recovery_code TEXT
  )
""")
```

Цей SQL-запит виконується один раз під час ініціалізації сервера або першого запуску.

Основні операції з базою:

1. Додавання нового користувача:

cursor.execute("INSERT INTO users (username, password_hash) VALUES (?, ?)", (username, hash))

2. Перевірка існування користувача:

cursor.execute("SELECT * FROM users WHERE username = ?", (username,))

3. Перевірка пароля (після вибірки хешу). Порівняння введеного пароля збереженим хешем (bcrypt.checkpw або аналог).

4. Оновлення прогресу гравця:

cursor.execute("UPDATE users SET level = ?, score = ? WHERE username = ?", (level, score, username))

5. Отримання профілю:

cursor.execute("SELECT username, level, score FROM users WHERE username = ?", (username,))

6. Видалення облікового запису (опціонально):

cursor.execute("DELETE FROM users WHERE username = ?", (username,))

Безпека роботи з даними

Щоб захистити базу:

– всі SQL-запити формуються з параметрами (placeholders), що запобігає SQL-ін'єкціям;

– паролі зберігаються в хешованому вигляді;

– на рівні SQLite можуть застосовуватись унікальні обмеження (наприклад, на username).

Переваги використання SQLite:

– Швидкий старт – не потребує додаткових налаштувань;

– Простота інтеграції з FastAPI – через sqlite3 або SQLAlchemy;

– Надійність для невеликих застосунків або MVP;

– Зручність перенесення між комп’ютерами – достатньо скопіювати один файл.

Обмеження:

– Обмежена масштабованість (не підходить для тисяч одночасних підключень);

– Не підтримує повноцінні транзакції з багатьох клієнтів одночасно;

– Не має системи користувачів і прав доступу, як PostgreSQL або MySQL.

У підсумку, SQLite повністю задовольняє потреби поточного проєкту: проста, легка і ефективна для зберігання облікових записів, ігрових рівнів і статистики. У разі масштабування гри її легко можна замінити на повноцінну СКБД(Система Керування Базами Даних) без зміни логіки API та перейти на онлайн бази з родини SQL.

3.3 Обробка запитів через FastAPI (ендпоїнти)

У серверній частині гри «Танковий бій» основну взаємодію між клієнтською частиною та сервером реалізовано через HTTP-ендпоїнти, які обробляються фреймворком FastAPI. Кожен ендпоїнт відповідає за виконання певної операції: авторизацію, реєстрацію, оновлення прогресу, отримання профілю тощо.

FastAPI дозволяє створювати ендпоїнти з чіткою типізацією параметрів, валідацією введених даних (через Pydantic) та зручним маршрутизуванням. Кожен маршрут супроводжується описом методів HTTP (GET, POST тощо) і автоматично документується через Swagger UI.

Загальна структура ендпоїнта у FastAPI:

– @app.post("/register") – декоратор, який вказує, що цей маршрут обробляє POST-запит за адресою /register;

– user: UserCreate – модель вхідних даних, описана за допомогою Pydantic.

Основні ендпоїнти, реалізовані у проєкті:

1. Реєстрація нового користувача POST /register

– Приймає логін і пароль;

– Перевіряє, чи користувач уже існує;

– Хешує пароль і додає новий запис у базу;

– Генерує унікальний id

– Повертає статус операції.

2. Авторизація (вхід) POST /login

– Перевіряє введені дані;

– У разі успіху – повертає профіль або токен (опціонально);

– У разі помилки – повертає 401 Unauthorized.

3. Отримання профілю користувача GET /profile/{username}

– Повертає дані користувача: логін, рівень, рейтинг;

– Працює лише для авторизованих користувачів (у майбутньому – з токенами).

4. Оновлення прогресу (після проходження рівня) POST /level/complete

– Приймає JSON-об’єкт із даними (рівень, рахунок, кількість зірок);

– Оновлює таблиці progress, progress_summary у SQLite;

– Повертає підтвердження збереження.

5. Вивід усіх користувачів (для leaderboard) GET /users

– Повертає список користувачів (обмежений набір полів);

– Може використовуватись для відображення таблиці лідерів.

Автоматична документація

FastAPI автоматично формує інтерактивну документацію для всіх маршрутів:

– Swagger UI: http://localhost:8000/docs

– ReDoc: http://localhost:8000/redoc

Це дозволяє зручно тестувати запити без потреби у сторонніх інструментах типу Postman.

Переваги архітектури ендпоїнтів FastAPI:

– Типізовані вхідні дані – менше помилок, краща безпека;

– Автоматичне логування та обробка помилок;

– Підтримка асинхронності – швидка відповідь навіть при високому навантаженні;

— Можливість легкої інтеграції нових функцій – наприклад, чату, PvP або API для адміністратора.

Таким чином, використання FastAPI для реалізації ендпоїнтів забезпечує структуровану, безпечну й масштабовану серверну архітектуру, яка зручно обслуговує клієнтські запити й дозволяє розширювати функціональність без порушення логіки проєкту.

3.4 Передача та оновлення даних між клієнтом і сервером

У проєкті гри «Танковий бій» взаємодія між клієнтською та серверною частинами реалізована через обмін HTTP-запитами у форматі JSON, що відповідає архітектурному стилю REST. Така модель взаємодії дозволяє клієнтській частині надсилати дані про дії гравця, а серверу — обробляти ці події, оновлювати стан гри та зберігати прогрес користувача у базі даних.

Основні напрямки обміну даними:

1. Надсилання запитів з клієнта (вгору):

– вхідні форми (реєстрація, логін);

– завершення рівня (статус, рахунок , кількість зірок);

– оновлення профілю або налаштувань.

2. Отримання даних із сервера (вниз):

– профіль гравця (рівень, рейтинг);

– доступні рівні;

– підтвердження дій або повідомлення про помилки.

Приклад типового обміну лістинг3.4.1

Лістинг 3.4.1 – обмін даними

```
      #Клієнт-сервер (після проходження рівня)
POST /level/complete
{
  "username": "Player1",
  "level": 3,
  "score": 1500,
  "status": "completed"
}
      #Сервер → клієнт (відповідь)
{
  "message": "Level progress saved successfully",
  "new_level": 4
}
```

Механізм оновлення даних у базі

Після отримання коректного запиту сервер:

1. Перевіряє авторизацію користувача.
2. Виконує SQL-запит до бази users.db:
3. Повертає підтвердження або повідомлення про помилку.

Таким чином реалізується односпрямована синхронізація прогресу.

Асинхронна передача даних

FastAPI підтримує асинхронні функції async def, що дозволяє:

– не блокувати обробку запитів;
– приймати запити одночасно від декількох клієнтів;

– швидко відповідати, навіть при пікових навантаженнях.

Кодування та структура даних

Усі дані передаються у форматі JSON з явною структурою:

– ключі мають зрозумілі назви (username, level, score);

– значення проходять перевірку через Pydantic;

– неприпустимі запити відхиляються зі статусом 400 Bad Request.

Взаємодія через JavaScript на клієнті

На фронтенді, запити до сервера надсилаються через функції типу fetch() або AJAX лістинг 3.4.2.

Лістинг 3.4.2

```
fetch('/level/complete', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    username: 'Player1',
    level: 3,
    score: 1500,
    status: 'completed'
  })
})
.then(res => res.json())
.then(data => {
  console.log(data.message);
});
```

Цей механізм дозволяє реалізувати динамічну взаємодію без перезавантаження сторінки.

Обробка помилок і відповіді

Сервер чітко сигналізує клієнту про результат запиту:

- успішна дія – статус 200 OK, повідомлення;
- помилка в даних – 400 Bad Request;
- відсутність користувача – 404 Not Found;
- помилка доступу – 401 Unauthorized.

Це дозволяє клієнтській частині гнучко реагувати на стан гри та дії користувача.

У підсумку, передача та оновлення даних між клієнтом і сервером реалізовані за допомогою асинхронних REST-запитів, що гарантує гнучкість, швидкодію та масштабованість серверної логіки гри.

ВИСНОВКИ

У ході виконання класифікаційної роботи було досягнуто поставленої мети розробити функціональну, безпечну та масштабовану серверну частину для веборієнтованої 2D-гри «Танковий бій». Робота охоплює повний цикл створення серверної інфраструктури: від аналізу вимог до реалізації API та тестування функціональності.

Основні результати роботи:

1. Аналіз вимог та проектування:

Визначено функціональні та нефункціональні вимоги до серверної частини гри.

– Обґрунтовано вибір технологій: Python, FastAPI, Uvicorn, SQLite та Pydantic.

– Спроєктовано REST-архітектуру з чіткою структурою API для взаємодії з клієнтами.

2. Реалізація серверної логіки:

– Реалізовано механізми реєстрації, авторизації та обробки запитів користувача.

– Створено модулі для збереження прогресу, роботи з базою даних, обробки профілів.

– Забезпечено надійну структуру запитів, валідацію вхідних даних і захист від SQL-ін’єкцій.

3. Тестування і перевірка працездатності:

– Проведено ручне тестування роботи ендпоїнтів через Swagger UI та браузерні запити.

– Перевірено коректність обробки помилок, авторизації та оновлення прогресу.

– Підтверджено стабільність і узгодженість між клієнтом і сервером.

Загальні висновки:

– Стабільність та надійність: Сервер успішно обробляє запити, захищає дані та забезпечує постійний доступ до функціоналу гри.

– Масштабованість: Завдяки використанню FastAPI та розділенню логіки, проєкт готовий до розширення і підтримки додаткових функцій.

– Безпека: Застосовано валідацію, захист від SQL-ін’єкцій та контроль помилок.

– Продуктивність: Завдяки асинхронному виконанню (Uvicorn), сервер швидко реагує на запити навіть при збільшенні навантаження.

– Перспективи розвитку:

– Інтеграція з клієнтською частиною для створення повноцінного ігрового середовища.

– Розширення функціональності: нові рівні, режими гри, рейтинги, мультиплеєр.

– Перехід до більш потужної бази даних (наприклад, PostgreSQL) у разі масштабування.

– Додавання токен-автентифікації, логування сесій, а також підключення HTTPS.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Martelli, A., Ravenscroft, A., Ascher, D. Python Cookbook. O'Reilly Media, 2022. 880 p.

2. Smith, J. Building Web APIs with Python and FastAPI. Packt Publishing, 2023. 320 p.

3. Sheifer, A. REST API Design. O'Reilly Media, 2019. 180 p.

4. Grinberg, M. Flask Web Development: Developing Web Applications with Python. O'Reilly Media, 2018. 258 p.

5. Chandramoulli, K. FastAPI for Beginners: Modern, Fast (High-Performance), Web Framework for Building APIs with Python 3.6+. Independently published, 2021. 150 p.

6. Stanislavskyi, V. I. Основи захисту інформації : підручник. Київ : Кондор, 2019. 312 с.

7. Stoltz, P. Learn SQL Quickly: A Beginner's Guide to Learning SQL, Even If You're New to Databases. Independently published, 2020. 220 p.

8. Holovaty, A., Kaplan-Moss, J. The Definitive Guide to Django: Web Development Done Right. Apress, 2017. 400 p.

9. FastAPI. High performance, easy to learn, fast to code and ready for production. – URL: https://fastapi.tiangolo.com (дата звернення: 14.01.2025).

10. Uvicorn. ASGI web server implementation for Python. – URL: https://www.uvicorn.org (дата звернення: 10.06.2025).

11. Pydantic. Data validation and settings management using Python type annotations. – URL: https://docs.pydantic.dev (дата звернення: 14.04.2025).

12. SQLite Documentation. – URL: https://www.sqlite.org/docs.html (дата звернення: 06.04.2025).

13. SQL Injection | OWASP Foundation [Електронний ресурс]. URL: https://owasp.org/www-community/attacks/SQL_Injection (дата звернення: 03.02.2025).

14. REST API Tutorial. [Електронний ресурс]. URL: https://restfulapi.net (дата звернення: 10.03.2025).