

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»

Завідувач кафедри

_____ Наталія АУШЕВА
“ ” _____ 2026 р.

**Дипломна робота
на здобуття ступеня бакалавр**

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Вебзастосунок планування задач з елементами ігрофікації та віртуальним аватаром”

Виконала: студентка 4 курсу, групи ТР-25

Білоус Олександра Володимирівна

(прізвище, ім’я, по батькові)

_____ (підпис)

Керівник: доцент Кафедри цифрових технологій в
енергетиці ІАТЕ кандидат економічних наук

Сегеда Ірина Василівна

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Рецензент доцент В.о. директора ННІ АТЕ КПІ ім.

Ігоря Сікорського, д.т.н., проф. Черноусенко О.Ю.

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Н.контроль Асистент Костянтин ЗДОР

(посада, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів без
відповідних посилань.

Студентка _____

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)

спеціальність 122 “Комп’ютерні науки”

Освітньо-професійна
програма

“Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ

Завідувач кафедри

Наталія АУШЕВА

(підпис)

” ” _____ 2026р.

ЗАВДАННЯ

на дипломну роботу студенту

БІЛОУС Олександрі Володимирівні

(прізвище, ім’я, по батькові)

1. Тема роботи “Вебзастосунок планування задач з елементами ігрофікації та віртуальним аватаром”

Науковий керівник Сегеда Ірина Василівна, доцент, кандидат економічних наук
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від ” ____ ” травня 2026 р. № _____

2. Термін подання студентом роботи “01” червня 2026 р.

3. Вихідні дані до роботи мови програмування C# TypeScript, середовища розробки Microsoft Visual Studio Code, Microsoft Visual Studio, середовище побудови діаграм DrawIO, мова розмітки HTML та стилів CSS

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі програмні рішення та можливі засоби реалізації взаємодії, обґрунтувати вибір програмних застосунків і шляхи розробки інформаційної системи, розробити програмне забезпечення, розробити користувацький інтерфейс для візуалізації роботи системи.

5. Орієнтований перелік ілюстративного матеріалу: зображення існуючих аналогів, DFD 0 та 1 рівнів, функціональна модель програми, алгоритм ігрового елемента, діаграма прецедентів, діаграма компонентів програми та клієнтської частини програми, діаграма пакетів серверної частини програми, концептуальна діаграма

бази даних, DTO-об'єкти, діаграма класів серверної частини, зображення готового програмного продукту.

6. Дата видачі завдання 10.09.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	10.01.2026	
2.	Вивчення та аналіз задачі	18.01 — 25.01.2026	
3.	Розробка архітектури та загальної структури системи	04.02 — 11.02.2026	
4.	Розробка структур окремих підсистем	12.02 — 16.02.2026	
5.	Програмна реалізація системи	22.02 — 20.04.2026	
6.	Оформлення пояснювальної записки	02.05 — 12.05.2026	
7.	Захист програмного продукту	13.05.2026	
8.	Передзахист	03.06.2026	
9.	Захист	15.06 — 19.06.2026	

Студент

(підпис)

Олександра БІЛОУС
(прізвище та ініціали)

Керівник роботи

(підпис)

Ірина СЕГЕДА
(прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 63 сторінок. 30 ілюстрацій, 1 додаток, 40 посилань. У роботі описано етапи розробки веб-додатку для планування задач з елементами ігрофікації.

Метою роботи є розробка вебзастосунку для підвищення мотивації користувача до виконання поставлених задач у задані проміжки часу через наявність системи віртуальних нагород.

У роботі використовувалися наступні засоби: мови програмування C# та TypeScript, мова розмітки HTML, мова стилів CSS, фреймворки Entity Framework та Angular, СУБД MsSQL Server. Основою архітектури додатку є принципи Single Page Application та Object-Relational Mapping.

В ході виконання поставленої задачі реалізовано вебдодаток-планер з системою винагородження та штрафів віртуальною валютою користувача, можливістю придбання віртуальних предметів та аксесуарів для облікового запису користувача.

Ключові слова: SPA, REST API, ігрофікація, планер, віртуальна нагорода.

ANNOTATION

The thesis consists of 63 pages, 30 illustrations, 1 appendices, and 40 references. The thesis describes the stages of developing a web application for task planning with gamification elements.

The goal of this thesis is to develop a web application designed to increase user motivation to complete assigned tasks within specified time frames through a system of virtual rewards.

The following tools were used in this thesis: the programming languages C# and TypeScript, the HTML markup language, the CSS stylesheet language, the Entity Framework and Angular frameworks, and the MsSQL Server database management system. The application architecture is based on the principles of Single Page Application and Object-Relational Mapping.

In the course of completing the assigned task, a web planner application was implemented, featuring a system of rewards and penalties using the user's virtual currency, as well as the ability to purchase virtual items and accessories for the user's account.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ	8
ВСТУП.....	11
1.1 Постановка задачі побудови вебзастосунку “AdvenTask”	13
1.2 Аналіз існуючих рішень	15
1.2.1 Додатки для фіксації запланованих завдань	15
1.2.2 Додатки з використанням ігрофікації.....	17
1.3 Проектування діаграми прецедентів та функціональної моделі вебзастосунку-планувальника “AdvenTask”	18
1.3.1 Контекстна діаграма (DFD Рівень 0) вебзастосунку -планувальника “AdvenTask”.....	19
1.3.2 Функціональна модель вебзастосунку (DFD Рівень 1).....	20
1.3.3 Діаграма прецедентів вебзастосунку	22
2 МЕТОДИ ТА ЗАСОБИ РОЗРОБКИ ВЕБЗАСТОСУНКУ-ПЛАНУВАЛЬНИКА “ADVENTASK”	25
2.1 Вибір засобів реалізації серверної частини вебзастосунку	25
2.2 Вибір засобів реалізації клієнтської частини вебзастосунку.	26
2.3 Вибір системи управління базами даних вебзастосунку	27
2.4 Вибір тестування додатку	27
2.5 Архітектурна побудова проекту	28
2.6 Структура серверної частини проекту	32
2.7 Архітектура клієнтської частини.....	33
2.8 Архітектура ігрової логіки вебзастосунку.....	36
2.9 Архітектура бази даних вебзастосунку	39
3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ	41
3.1 Програмна реалізація серверної частини	41
3.1.1 Реалізація об’єктної моделі та доступу до даних	41
3.1.2 Шар сервісів та реалізації бізнес-логіки зарахування нагороди.....	52
3.1.3 Шар представлення. Реалізація REST API контролерів	53
3.1.4 Конфігурація серверного середовища, впровадження залежностей	55
3.1.5 Модульне тестування серверної логіки	56
3.2 Програмна реалізація клієнтської частини.....	57

3.2.1 Управління станом через Angular Signals	58
3.2.2 Шар системної логіки та інфраструктури клієнтської частини додатку .	59
3.2.3 Функціональні модулі	60
3.2.4 Візуальне оформлення.....	61
4. РОБОТА КОРИСТУВАЧА З ПРОГРАМОЮ	62
ВИСНОВКИ.....	66
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	68
ДОДАТОК А.....	72
ДОДАТОК Б	84

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Ігрофікація (гейміфікація) – впровадження використання ігрових механік та елементів у неігрових контекстах для підвищення залученості та мотивації користувачів.

SPA (Single-Page Application) – архітектурний підхід розробки вебзастосунків, згідного якого весь додаток розміщується на одній сторінці HTML. Контент на сторінці оновлюється динамічно та не потребує перезавантаження сторінки під час взаємодії користувача з додатком.

REST API (Representational State Transfer Application Programming Interface) – архітектурний стиль взаємодії компонентів системи через протокол HTTP, заснований на передачі станів представлень.

Тайм-менеджмент – сукупність технік та методів раціональної організації робочого та особистого часу.

DTO (Data Transfer Object) – архітектурний патерн, що представляє структуру даних, яка позбавлена бізнес-логіки, але виступає компонентом для обміну інформацією між шарами системи.

RPG (Role-Playing Game) – жанр у відеограх, в яких головною характеристикою є наявність системи розвитку характеристик та навичок.

ORM – технологія в програмування, що дозволяє передавати дані між об'єктами об'єктно-орієнтованих мов програмування та записами в реляційних базах даних. Дозволяє уникнути необхідності написання нагромадженного коду для доступу до даних, забезпечує автоматичну генерацію SQL-запитів та міграцію схеми баз даних з написаного коду.

UX (User Experience) – термін, що описує сукупність досвіду, вражень та емоцій користувача під час взаємодії з програмним продуктом.

DFD (Data Flow Diagram) – діаграма потоків даних; графічна модель, що описує рух інформації між процесами та сховищами в системі.

JWT (JSON Web Token) – відкритий стандарт для створення токенів доступу, що використовуються для безпечної ідентифікації та передачі інформації про користувача.

CRUD (Create, Read, Update, Delete) – перелік базових операцій для роботи з даними: створення, читання, оновлення та видалення.

UML (Unified Modeling Language) – уніфікована мова графічного опису для моделювання, візуалізації та документування архітектури програмних систем.

HTTP / HTTPS (HyperText Transfer Protocol / Secure) – основні мережеві протоколи передачі даних, що підтримують шифрування (у випадку HTTPS).

LINQ (Language Integrated Query) – вбудована технологія платформи .NET, яка дозволяє виконувати запити до даних безпосередньо за допомогою мови програмування C#.

ORM (Object-Relational Mapping) – технологія в програмуванні, що дозволяє передавати дані між об'єктами мови програмування та записами в реляційних базах даних без написання прямого SQL-коду.

СУБД – система управління базами даних; програмне забезпечення для створення, редагування та керування структурою БД (у роботі використано Microsoft SQL Server).

ACID (Atomicity, Consistency, Isolation, Durability) – набір властивостей (атомарність, узгодженість, ізолюваність, довговічність), що гарантують надійність виконання транзакцій у базах даних.

DRY (Don't Repeat Yourself) – принцип розробки програмного забезпечення, спрямований на уникнення дублювання коду шляхом винесення спільної логіки в окремі модулі або сервіси.

Signals (Angular Signals) – система реактивності у фреймворку Angular, що дозволяє автоматично відстежувати залежності та виконувати гранулярне оновлення інтерфейсу.

JSON (JavaScript Object Notation) – легкий текстовий формат обміну даними, заснований на парах «ключ-значення».

Десеріалізація – процес відновлення структури об'єкта (перетворення тексту формату JSON у функціональний об'єкт мови програмування) для подальшої обробки його властивостей.

3NF – третя нормальна форма; стан структури бази даних, при якому виключено надмірність інформації та забезпечено її логічну цілісність.

GUID (Globally Unique Identifier) – унікальний 128-бітний ідентифікатор, що використовується як первинний ключ для захисту від несанкціонованого перебору записів.

CORS (Cross-Origin Resource Sharing) – механізм безпеки, що дозволяє вебзастосунку отримувати доступ до ресурсів іншого домену, відмінного від того, з якого його було завантажено.

ВСТУП

В умовах сучасності життя людей зазнає вплив стрімкого темпу життя та зростання обсягу інформаційного простору, в якому необхідно перебувати щодня та опрацьовувати величезні обсяги інформації, питання ефективного планування та раціонального розподілу часу стає відчутно важливим для підтримки продуктивності. Постійна зміна пріоритетів та необхідність постійного утримання в пам'яті певного об'єму завдань вимагають не тільки дисципліни, а й гнучкого інструменту. Наявні традиційні методи планування завдань часто стикаються з проблемою швидкої втрати мотивації та уваги користувачів. Відчуття відсутності досягнутого прогресу та рутинність процесу заповнення однотипових нотатків стає відчутним недоліком для користувачів.

Впровадження механізму ігрофікації в додатки-планувальники є одним з шляхів подолання описаних проблем, оскільки дозволяє урізноманітнити досвід користування планерів та заохочує користувачів дотримуватися поставлених задач [1]. Використання елементів гри, таких як отримання очок досвіду та віртуальної валюти чи предметів, обрано через на систему психологічної підтримки мотивації користувача, що перетворює рутинні процеси на унікальний інтерактивний досвід [2].

Метою дипломної роботи є проєктування та розробка інтерактивного програмного забезпечення для планування завдань, з використанням механік ігрової прогресії та винагородження віртуальними грошовими одиницями та очками досвіду. Система має реалізовувати конвертацію успішно виконаного завдання у цифрові активи, забезпечити систему втрати частини нагороди за порушення заданих часових проміжків та механізм придбання віртуальних прискорювачів росту та колекційних аватарів для облікового запису користувача.

Завданням даної роботи є проведення огляду та аналізу існуючих додатків-планувальників та рішень з елементами ігрофікації, виявити недоліки та переваги; визначити функціональні та нефункціональні вимоги до системи; обґрунтувати обраний стек технологій розробки; спроектувати архітектуру системи: базу даних,

модулі обробки бізнес-логіки додатку; реалізувати алгоритми керування завданнями, роботи магазину, дій покращень та штрафів; провести тестування розробленого програмного комплексу та з урахуванням виявлених недоліків внести зміни в програмний код, випробувати розроблений програмний продукт на реальних користувачах.

Результати проведеної роботи були представлені на студентській конференції “Сучасні проблеми наукового забезпечення енергетики” в тематичній секції конференції “Інформаційні технології та комп'ютерне моделювання” (Доповідь “Розробка веб-застосунку для планування задач з елементами ігрофікації та віртуальним аватаром”, квітень 2026р.).

1 ЗАДАЧІ ПОБУДОВИ ВЕБЗАСТОСУНКУ “ADVENTASK” ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Об’єктом розробки в даній дипломній роботі є програмне забезпечення з управління персональними завданнями та інтегрованим елементом гейміфікації. Головна ідея додатку полягає в побудові алгоритму, який конвертує продуктивність користувача у виконанні поставлених задач в віртуальні винагороди.

1.1 Постановка задачі побудови вебзастосунку “AdvenTask”

Основна проблема систем індивідуального менеджменту - низький рівень утримання уваги користувачів. Більшість засобів планування виключно фокусуються на фіксації внесених записів щодо завдання. Такий підхід хоч і забезпечує виконання базового функціональності, проте з часом виникає відчуття рутинності, через це, введення тайм-менеджменту сприймається як додаткове навантаження, а не допоміжний засіб. Для подолання цього бар'єру необхідно впроваджувати механізми формування стійкої звички через систему винагород [3].

Щоб вирішити визначену проблему, в даній роботі поставлене завдання - спроектувати та розробити програмний комплекс, який використовує концепцію залучення в повсякденні рутинні справи елементи гри. Головна ідея полягає в створенні наступного алгоритму: виконане завдання - отримання нагороди з урахуванням складності та побічних впливаючих факторів (наявності модифікаторів або штрафів) - покращення ігрового профілю користувача.

Вимоги до системи є специфічними через необхідність поєднання двох різних середовищ: чітко структурованих завдань та гнучка ігрова економіка. Щоб забезпечити комфортний досвід використання додатка, вміст застосунку не має містити перевантаження ігровими елементами, а сама логіка ігрового аспекту(операції з ігровою валютою) має бути чітко математично обґрунтована,

не містити схильності в сторону надмірної важкості або легкості. Програмний продукт має наочно демонструвати паралель між продемонстрованою “продуктивністю” та результатом - накопиченим капіталом і здобутими віртуальними активами. Система кастомізації аватара та придбання модифікаторів прогресу в даному випадку виступатиме засобом візуалізації досягнутого прогресу.

Враховуючи, що застосунок призначений для щоденного використання, ПЗ має бути реалізовано за принципом SPA, мати мінімальну кількість завантажень, швидке опрацювання запитів клієнтів та мати мінімальну кількість кроків до цільової дії[4].

З огляду на зазначене вище, функціональні та нефункціональні вимоги сформовані наступним чином.

Основна функціональна вимога до додатку - функціонал, що дозволяє оперувати блоком з завданням: створювати, редагувати та визначати класифікацію й надавати пріоритезацію. Алгоритм має автоматично підраховувати та нараховувати очки досвіду та віртуальну валюту. Розрахунок здійснюється з урахуванням вхідних параметрів, вказаних користувачем(складність), застосованого в період виконання задачі модифікатора і наявності або відсутності покарання за недотримання вказаних на момент створення чи редагування дедлайнів. Система має містити модуль, що відповідати за логіку роботи внутрішньо ігрового магазину. В секції магазину користувачі можуть придбати покращувачі для зростання обсягу здобутих ресурсів і косметичні модифікації для профілю користувача.

До нефункціональних вимог даного застосунок відносяться забезпечення стабільності роботи додатку як Single Page Application(SPA). Система має демонструвати швидкість відгуку на запити користувача та мати мінімізоване навантаження на систему як з клієнтської сторони, так і зі сторони надання послуги. Додаток-планувальник має забезпечувати цілісність даних, безпеку паролів, а також бути придатним до майбутніх змін, наприклад оновлення наявних механізмів, або ж додавання нового функціоналу.

1.2 Аналіз існуючих рішень

На сьогоднішній день на ринку програмних продуктів представлено досить широкий спектр продуктів, що надають послуги управління завданнями: класичні мінімізовані додатки-планери й складні системи для організації та тайм менеджменту, які застосовуються не тільки к персональні інструменти самоорганізації, а й як інструменти контролю для цілих корпоративних систем. Для розробки власного проєкту було проаналізовано рішення, що використовують різні підходи до мотивації користувача.

1.2.1 Додатки для фіксації запланованих завдань

Найближчим аналогом за концептуальною ідеєю є програмне забезпечення Habitica. Додаток представлений як кросплатформний застосунок, що має відкритий вихідний. Програмне забезпечення застосовує активне залучення користувача до ігрових процесів. Habitica реалізовує механіки RPG-ігор, серед яких: наявність “персонажа” що отримує пошкодження, якщо завдання не виконане, накопичує досвід та золота за успішну діяльність, наявний внутрішньо ігровий магазин, в якому можна купити різноманітне спорядження та залучення користувачів до виконання завдань, розрахованих на групу людей.[5]

Планувальник містить насичений аспект соціальної комунікації та співпраці, проте має надмірне наповнення інтерфейсу деталями та так званим “візуальним шумом”, необов’язковими механіками. Перелічені пункти можуть посприяти відволіканню користувача від першочергових цілей. На рисунку 1.2.1 приведено приклад вигляду додатку.

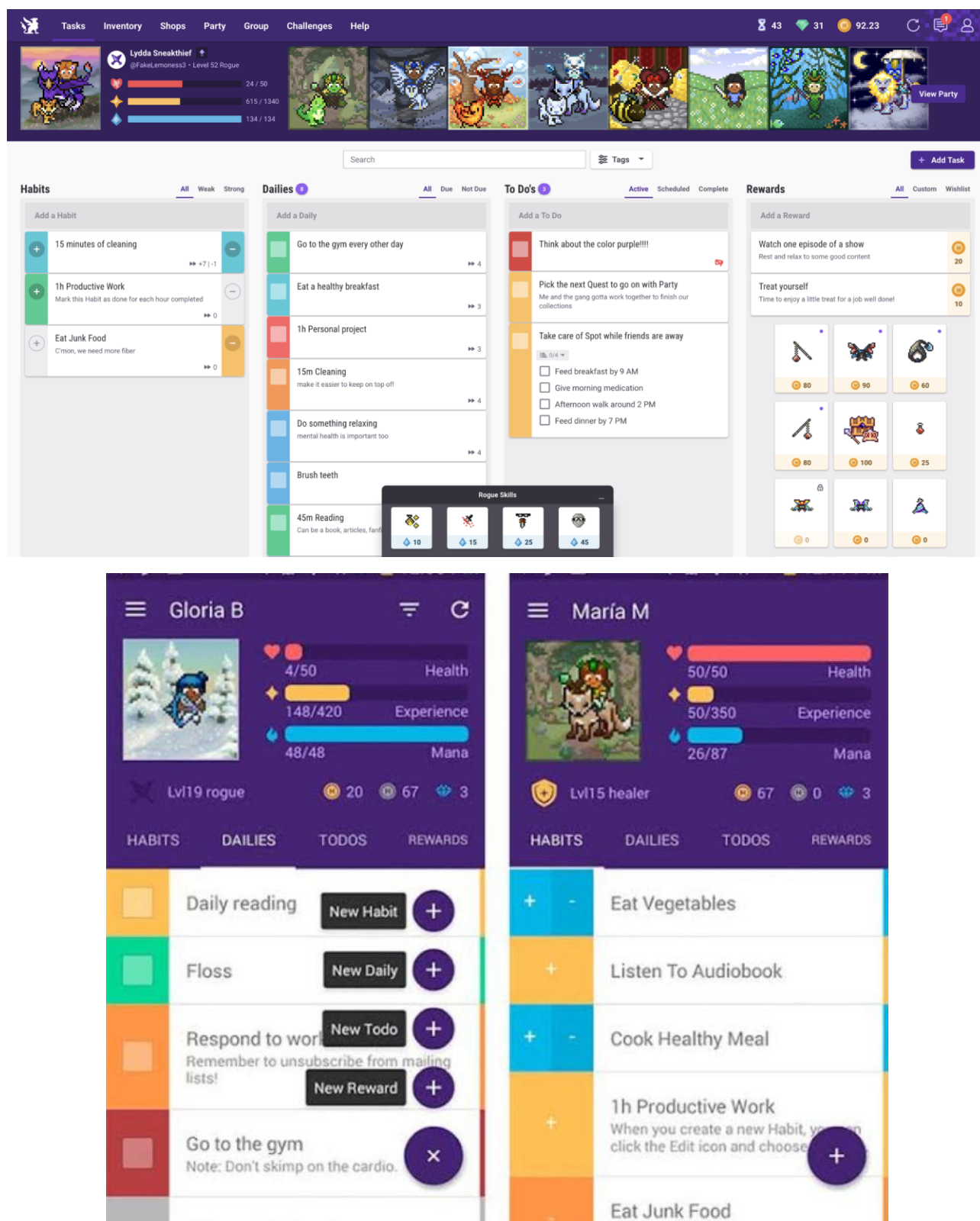


Рисунок 1.1 - Приклад інтерфейсу Habitica

Todoist - архітектурна та візуальна протилежність попередньому рішення. Додаток має мінімалістичний дизайн, якісний UX та вважається швидким в роботі[6]. Дизайн додатку зображено на рисунку 1.2.

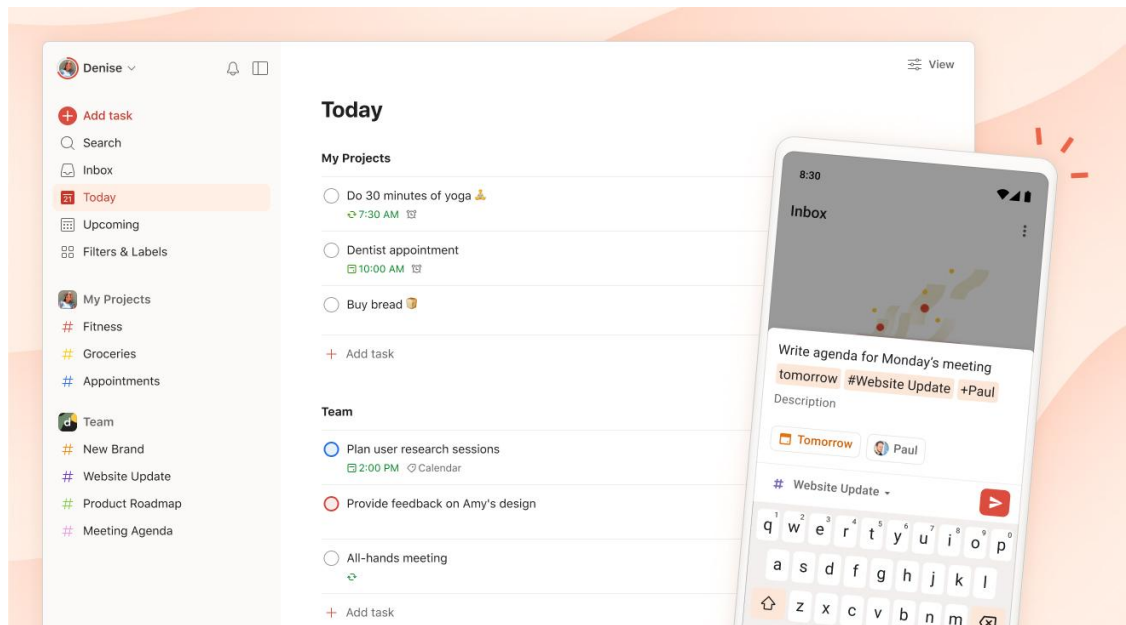


Рисунок 1.2. - Інтерфейс Todoist

Єдиний механізм “ігрофікації” - наявність системи “Карми”. За виконання завдань - користувач отримує бали пролуктивності та втрачає їх, коли завдання не є виконаним у вказані часові межі. Практичного використання дані бали не мають і слугують тільки для візуалізації статусу профілю. Додаток чітко виконує поставлену перед планувальником задачу - фіксує терміни та основні параметри задачі, проте ніяк не сприяє їх виконанню.

1.2.2 Додатки з використанням ігрофікації

Окремо слід зауважити вдале поєднання корисного додатку з ігровими елементами - Duolingo. Інтерфейс додатку відображено на рисунку 1.3. Хоча згадана платформа не є прямим інструментом планування задач, воно є відомим прикладом застосування ігрових механік для формування корисних звичок з вивчення іноземних мов[7].

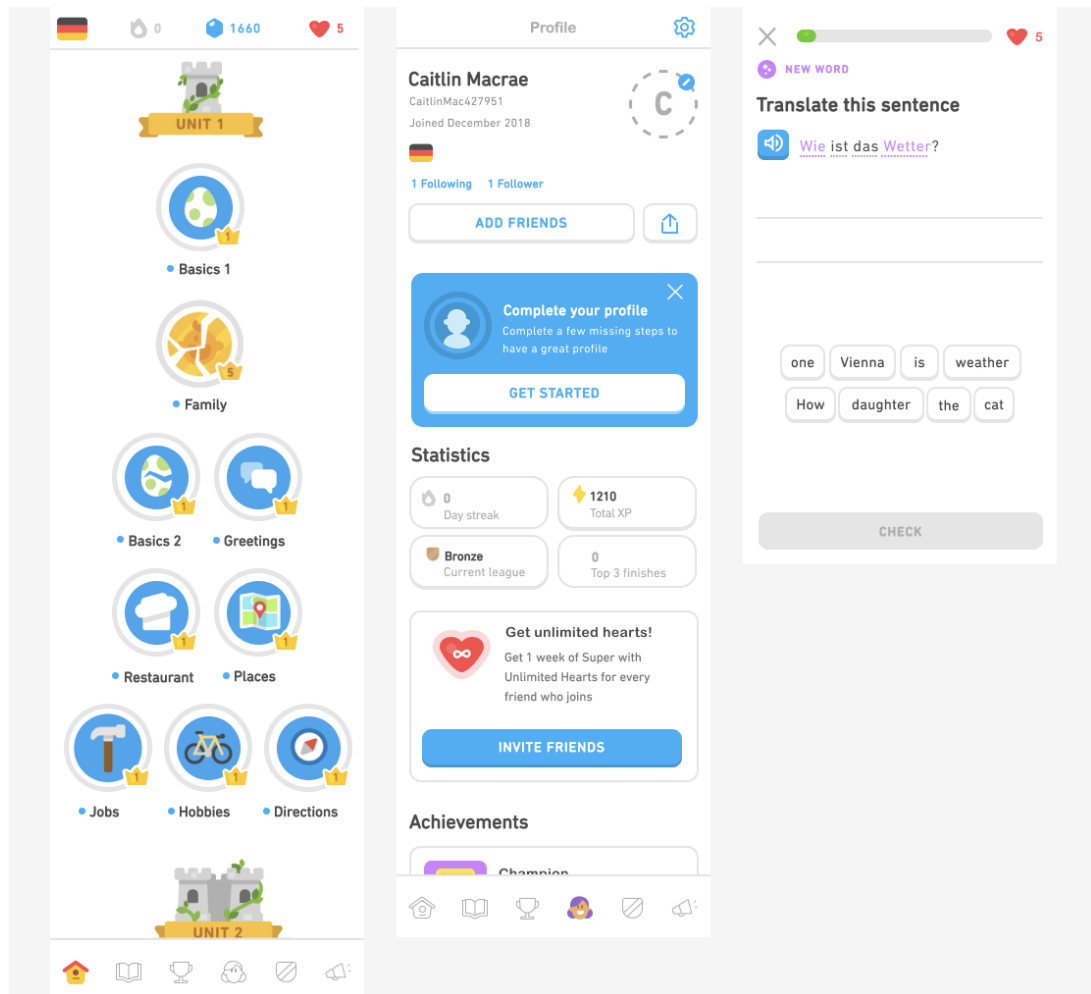


Рисунок 1.3 - Інтерфейс Duolingo

Система має механіки пов’язані на підтримці активності користувачів через “дні активності” та візуалізації досягнутого прогресу, використовуючи віртуальні медалі, звання та бали. Додаток не має візуального перевантаження, загальний дизайн є простим та лаконічним, тому користувачі не губляться та легко вивчають можливості додатку.

1.3 Проектування діаграми прецедентів та функціональної моделі вебзастосунку-планувальника “AdvenTask”

Ефективність реалізації будь-якого програмного продукту залежить від детального етапу проектування. Мають бути описані взаємодії користувачів з системою. Початковим етапом проектування будуть описані моделі системи

методологією DFD на двох рівнях(контекстна та функціональна декомпозиції) та діаграма прецедентів.

1.3.1 Контекстна діаграма (DFD Рівень 0) вебзастосунку - планувальника “AdvenTask”

Контекстна діаграма(рисунок 1.4) відображає найбільш узагальнений погляд на функціонал системи.



Рисунок 1.4 - Контекстна модель вебзастосунку

Основною зовнішньою сутністю системи є Користувач. Вони ініціюють вхідний потік даних, який складається з облікових даних до персоналізованого простору додатку, параметри запланованого завдання, запити на придбання нових елементів.

Вихідний потік даних включає отримання зароблених ресурсів, підтвердження оброблених запитів, відображення придбаних предметів та аватарів.

1.3.2 Функціональна модель вебзастосунку (DFD Рівень 1)

Функціональна модель описує внутрішні процеси застосунку, взаємодію між програмними модулями та сховищами даних. Методологія DFD дозволяє чітко розмежувати процеси обробки інформації та зони відповідальності кожного з сервісів [8]. Побудова моделі базується на принципах структурного аналізу, де система декомпонується на функціональні вузли, що перетворюють вхідні потоки даних у вихідні [9].

Вхідними даними виступають параметри, які визначають самі користувачі(дані облікового запису, параметри запису завдання). Процеси обробки охоплюють збереження записів до бази даних, їх зміну, видалення, а також обчислення винагород. Вихідними даними системами є результати запитів користувачів, до яких відноситься зміна даних профілю, балансу віртуальної валют чи досвіду та зміна параметрів завдання.

В діаграмі застосовують наступні позначення:

Зовнішня сутність - певна сутність(людина чи об'єкт), яка здатна виробляти або отримувати дані. Зображуються у вигляді звичайного прямокутника.

Сховище даних - сукупність даних, що знаходиться в стані спокою. Надає інформацію. Зображується за допомогою паралельних ліній.

Процес - певна діяльність, яка оперує даними та виконує бізнес-ціль. Зображують як прямокутник з заокругленими кутами.

Діаграма першого рівня деталізує внутрішній перебіг процесів програми, демонструючи взаємодію між модулями ПЗ та сховищами даних (рисунок 1.5).

Дана система розподілена на одну зовнішню сутність(Користувача), чотири функціональні блоки та три сховищ даних: Users(D1), Quests(D2), Market(D3).



Рисунок 1.5 - Функціональна модель програми

До чотирьох функціональних блоків відносяться:

1. Сервіс автентифікації
2. Управління завданнями
3. Шар ігрової логіки
4. Магазин та кастомізація

Сервіс автентифікації забезпечує авторизацію та автентифікацію користувачів та генерацію JWT-токенів для підтримки сесії користувача. Взаємодіє з сховищем даних користувачів(D1).

Сервіс управління завданнями реалізує логіку планування задач. Через операції CRUD користувач здійснює організацію та пріоритезацію задач. Здійснює операції над сховищем даних завдань(D2).

Шар ігрової логіки є головним модулем системи, що інтегрує результати планування з ігровою економікою. При статусі завершеного завдання, процес обчислює обсяг зароблених ігрових ресурсів, згідно до заданої складності.

Якщо заданий дедлайн був порушений, шар враховує це та зменшує обсяг нагороди, а за наявності модифікаторів - збільшує. Даний шар взаємодіє вже з двома сховищами даних - користувачів і магазину предметів(D1 та B3).

Останній сервіс - магазин та кастомізація. Даний шар відповідальний за реєстрацію вже придбаних товарів. Також взаємодіє з двома сховищами даних - користувачів та магазину.

Три сховища даних відповідають за збереження інформації про профілі користувачів, описів завдань та облік і характеристики предметів гри відповідно.

1.3.3 Діаграма прецедентів вебзастосунку

Для моделювання функціональних вимог і загального розуміння ролей та їх взаємодії між собою використовується діаграма прецедентів(діаграма варіантів використання) [10]. Діаграма прецедентів зображена на рисунку.1.6.

Основним актором вебзастосунку є Користувач. Він є ініціатором всіх прецедентів через взаємодію з інтерфейсом вебдодатку. Вторинний актор - Система, що представляє серверну частину проєкту. Він відповідає за виконання бізнес-логіки, валідацію та ведення записів в базі даних.

Функціонал системи можна розділити на наступні логічні блоки:

Управління обліковим записом. Користувач може авторизуватися або автентифікуватися для отримання доступу до вебдодатка. Обидві дії потребують вводу даних для створення запису або для знаходження вже зареєстрованого. Користувач має можливість змінити пароль до облікового запису, або ж, за наявності потреби - видалити профіль. Система опрацьовує ці запити, оперуючи над даними в базі даних.

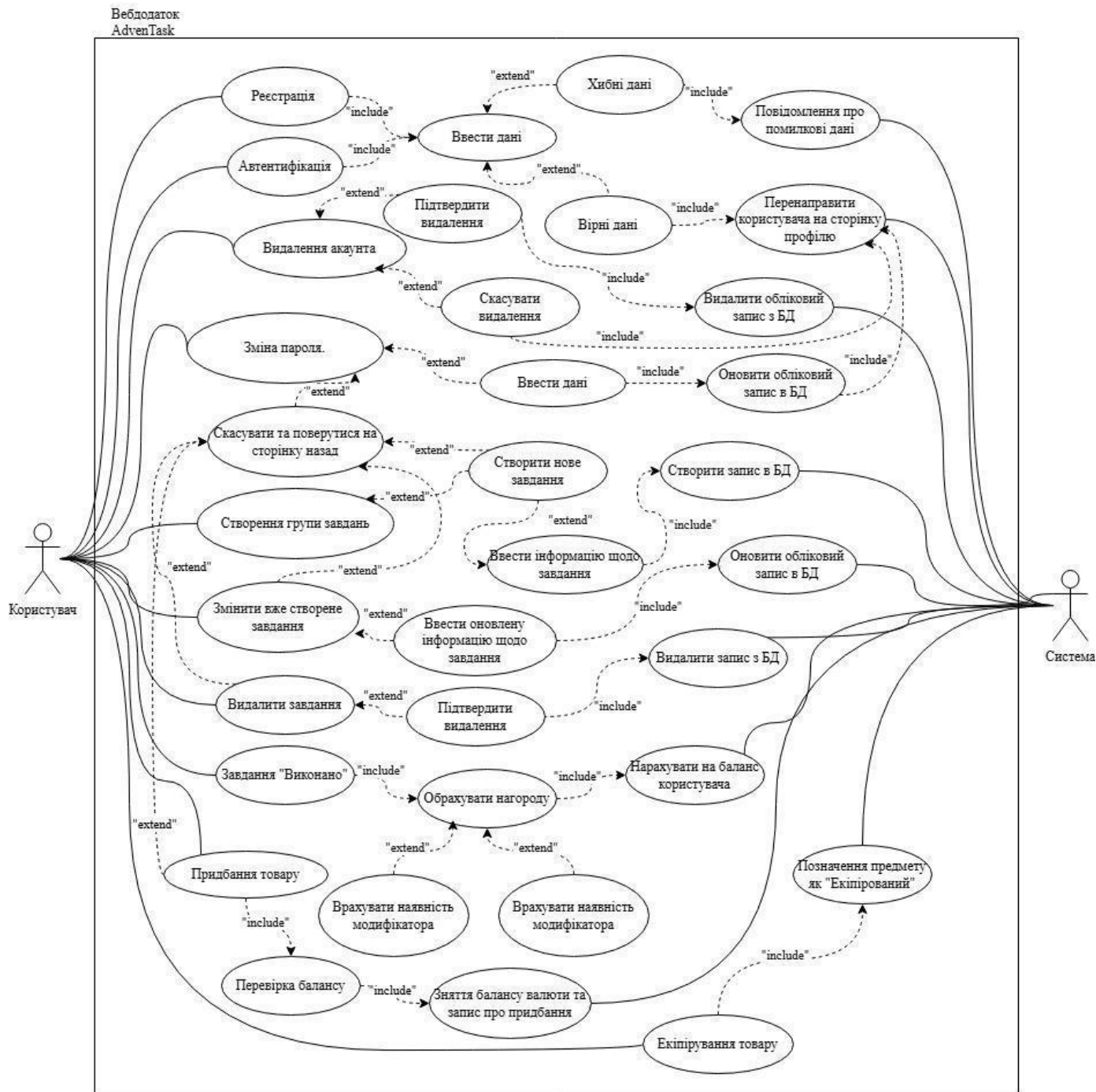


Рисунок 1.6 - Діаграма прецедентів AdvenTask

Управління завданнями. Перший крок - створити об'єкт, над яким надалі будуть проводитися дії. Спочатку створюється група "Квестів", всередині якої створюються вже окремі самостійні завдання. Користувач має повний контроль над циклом життя як списків завдань, так і окремими задачами - може створити додаткові, редагувати існуючі та видаляти існуючі.

Ігрова логіка. При присвоєнні завданню статусу “Виконано”, система обраховує обсяг зароблених ресурсів. За наявності модифікаторів-артефактів, згідно умови та параметрів артефакту система нараховуватиме додатковий обсяг ресурсів до балансу. Якщо ж завдання було виконане поза межами поставленого дедлайну - бонусні нарахування не здійснюються.

Придбання нових товарів та їх застосування. З моменту створення нового облікового запису, користувач має один стандартний аватар та жодного допоміжного модифікатора. При недостатній кількості валюти купівля неможлива. При придбанні артефакт з’являється в інвентарному просторі користувацького профілю, готовий до активації. Такі самі дії відбуваються із купівлею аватаром.

2 МЕТОДИ ТА ЗАСОБИ РОЗРОБКИ ВЕБЗАСТОСУНКУ-ПЛАНУВАЛЬНИКА “ADVENTASK”

Проектування системи базується на використанні поширеного та продуктивного стеку інструментів розробки, що дозволить зручну підтримку програмного продукту в майбутньому. Вибір в сторону систем з готовими стандартизованими рішеннями для різних задач також дозволяє зосередитися на реалізації унікальних механік проєкту.

2.1 Вибір засобів реалізації серверної частини вебзастосунку

Як основну платформу розробки серверної частини обрано .NET з мовою написання коду - C#. Даний вибір був зроблений з огляду на те, що платформа базується на кросплатформенних вебсерверах Kestrel, що показують високу продуктивність. Платформа .NET версії 10.0 надає розширені можливості для розробки високонавантажених систем, підтримує протоколи HTTP/S та безшовну інтеграцію з компонентами екосистеми Microsoft [11].

Використання мови C# дозволяє застосовувати принципи об'єктно-орієнтованого програмування для точного моделювання сутностей предметної області (завдань, артефактів, користувачів). Важливою перевагою є використання технології LINQ (Language Integrated Query), яка дозволяє уніфіковано та ефективно опрацьовувати дані з різних джерел, включаючи сервери баз даних та колекції в пам'яті [12].

Взаємодія з базою даних реалізована через Entity Framework Core — сучасний інструмент класу ORM (Object-Relational Mapping). Робота ведеться за принципом Code First, де схема бази даних автоматично генерується та оновлюється через механізм міграцій на основі програмних моделей даних [13]. Для забезпечення безпеки та керування доступом використано фреймворк

ASP.NET Core Identity. Він реалізує захист даних, включаючи готові алгоритми хешування паролів та механізми захисту від атак методом підбору (Brute-force). Як інструмент передачі даних між клієнтом та API обрано стандарт JWT (JSON Web Token)[14]. Даний стандарт відповідає специфікації RFC 7519[15]. Це дозволяє реалізувати автентифікацію без збереження стану сесії на сервері (stateless), що є критично важливим для продуктивності SPA-додатків.

Архітектура сервера побудована за принципом розмежування зон відповідальності (N-tier architecture), що відображено у структурі рішення: розділення на ядро, інфраструктуру та рівень API [16].

Сервера частина реалізована в середовищі програмування Visual Studio 2026.

2.2 Вибір засобів реалізації клієнтської частини вебзастосунку.

Клієнтська частина реалізована на фреймворку Angular. Даний фреймворк дозволяє створювати складні додатки, користуючись принципом SPA (Single Page Application). Мовою програмування обрано TypeScript, що забезпечує сувору типізацію даних та мінімізує кількість логічних помилок на етапі розробки [17].

Для забезпечення реактивності та точного контролю стану додатка застосовуються Angular Signals. Цей механізм дозволяє системі ефективно відслідковувати залежності між даними та виконувати гранулярне оновлення інтерфейсу без необхідності перевірки всього дерева компонентів. Це значно підвищує продуктивність застосунку, особливо в модулях динамічного відображення ігрових ресурсів та статусу квестів [18].

Взаємодія із серверною частиною здійснюється через вбудований модуль HttpClient. Для забезпечення безпеки та підтримки авторизованого доступу в

системі реалізовано механізм Interceptors, який автоматично додає JWT-токен до кожного HTTP-запиту, гарантуючи цілісність сесії користувача [19].

Візуальна складова клієнтської сторони додатка реалізована з використанням інтеграції спеціалізованих CSS-бібліотек NES.css. Дана бібліотека адаптує стандартні елементи інтерфейсу (кнопки, контейнери, діалогові вікна) під стилістику 8-бітної піксельної графіки. Вибір цього рішення обумовлений концепцією гейміфікації проєкту, оскільки такий стиль створює унікальну атмосферу RPG-гри та покращує візуальний досвід користувача [20].

Для об'єднання обраних засобів було обрано середовище програмування Visual Studio Code.

2.3 Вибір системи управління базами даних вебзастосунку

Для забезпечення надійного зберігання інформації, підтримки цілісності даних та реалізації транзакційної логіки було обрано реляційну систему управління базами даних Microsoft SQL Server. Вибір даного рішення зумовлений високими показниками продуктивності та повною підтримкою стандартів ACID. [21].

СУБД має інтеграцію із середовищем розробки Visual Studio через інструментарій SQL Server Object Explorer, що дозволяє взаємодіяти зі схемою даних, виконувати запити та проводити відлагодження бази даних безпосередньо в межах IDE.

2.4 Вибір тестування додатку

Проведення тестування є важливим аспектом розробки та підтримки програмного комплексу. Для цього кроку було впроваджено тестування, що базується на модулях тестування. Тести реалізуються для перевірки основних

бізнес-процесів, тому вирішено, що ця частина проєкту також буде розроблятися в IDE серверної частини з застосуванням фреймворку xUnit, що ж сучасним стандартом для раніше обраної платформи .NET. Фреймворк є гнучким, підтримує паралельне виконання тестів [22].

2.5 Архітектурна побудова проєкту

При побудові архітектури програмного продукту було обрано принципи Чистої архітектури, також відомої як Цибулева архітектура. Основа даного принципу - спрямувати залежність компонентів системи до її ядра, а кожен шар може взаємодіяти лише з тим рівнем, що знаходиться під ним. Схематичне зображення описаного принципу приведено на рисунку 2.1.

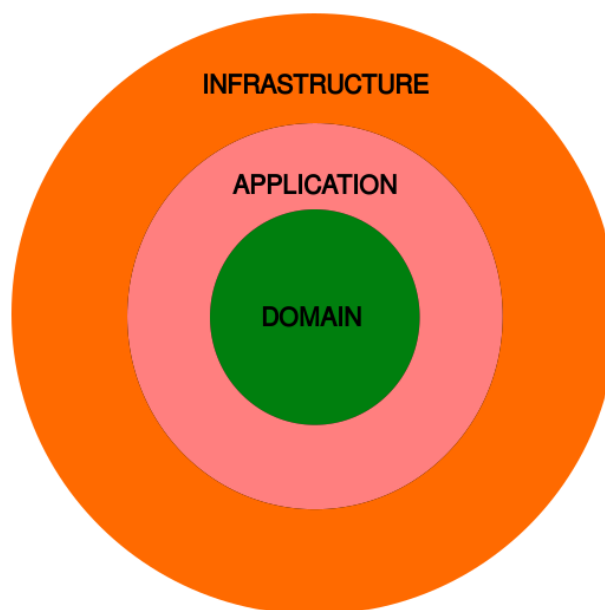


Рисунок 2.1 - Модель Чистої інфраструктури

Згідно з методологією Роберта Мартіна, такий підхід дозволяє відокремити бізнес-логіку від технічних деталей (баз даних, фреймворків, інтерфейсів) [23].

Структура розробленого рішення поділена на три основні рівні:

1. Core (Domain Layer) - рівень, який містить сутності та правила системи, відповідно є незалежним від інших.
2. Application - рівень, що реалізує доступність до сховища даних(бази даних) та зовнішніх служб. Містить сценарії використання даних та координує взаємодію між зовнішніми факторами та ядром.
3. Infrastructure (WebAPI) - рівень, що є зовнішнім інтерфейсом програми для реалізації взаємодії користувача з кодом системи.

Даний принцип забезпечує гнучкість системи. При зміні одного з компонентів не потребується переписування ключових моментів інших частин проекту.

Система побудована як поєднання Single Page Application(SPA) для клієнтської частини та REST API для серверної частини. Поєднання цих принципів забезпечує високу швидкість відгуку на запити користувача.

Схематично архітектуру представлено діаграмою компонентів UML на рисунку 2.2, яка описує взаємодію між окремими вузлами системи [10].

Клієнтська частина вебдодатка-планувальника реалізована за принципом SPA та організована як набір модульних компонентів та сервісів, де центральним вузлом комунікації між частинами проєкту є JWT Interceptor. Блок UI Components містить основні інтерфейси вебдодатку(Quests, Shop, Hero). Вони відповідають за відображення списків задач, ігрового магазину та профілю користувача. Взаємодія між ними та зміна сторінок без перезавантажень організовується компонентом Angular Router. Angular Services відповідно вміщає сервіси(QuestService, AuhService, ArtifactService) забезпечує формування запитів та управлінням станом даних на стороні клієнта та інкапсуляцію логіки взаємодії з сервером. JWT Interceptor забезпечує автоматичну ін'єкцію токенів авторизації у всіх запити до системи, які надходять від клієнтів до серверної частини.

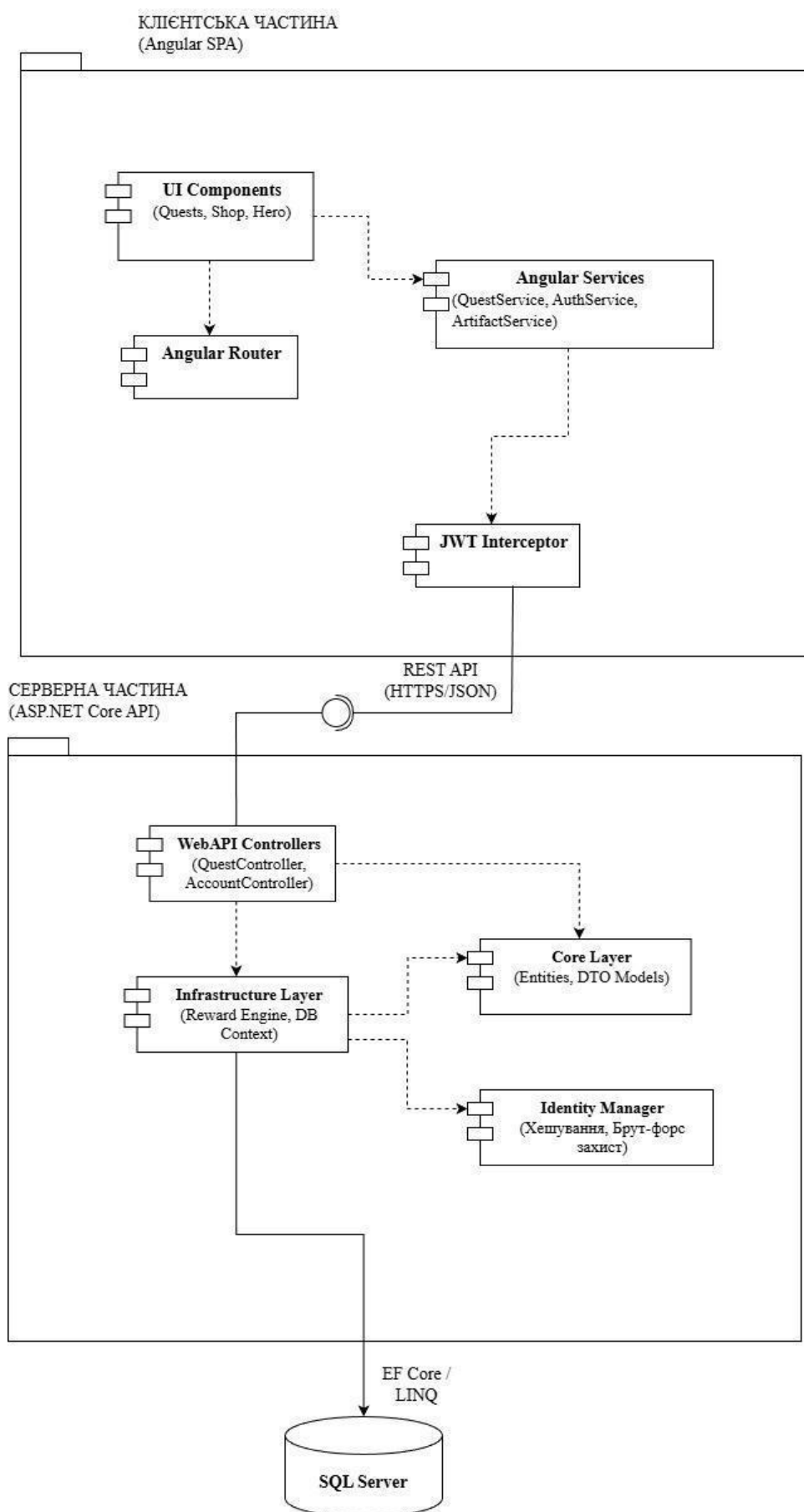


Рисунок.2.2 - Діаграма компонентів системи програмного комплексу

Серверна частина реалізує бізнес-логіку. Взаємодія між клієнтом та сервером відбувається через обмін JSON-об'єктів захищеним протоколом HTTPS. Даний шар через компонент WebApi Controllers(QuestController, AccountController) приймають виклики ззовні та делегують виконання завдань нижчому рівню - рівню інфраструктури(Infrastructure Layer).

Infrastructure Layer вміщує інформацію про технічні деталі. Тут зосереджені математичні алгоритми нарахування ігрових ресурсів(досвіду та монет), а також DB Context - керування підключенням до бази даних, Entity Framework Core для LINQ-запитів.Identity Manager - допоміжний модуль, що хешує паролі , керує профілями та забезпечує механізм захисту від підбора паролів.

Core Layer - рівень ядра системи, що є незалежним і вміщує моделі даних. Відповідно, взаємодія з базою даних інкапсульована з використанням фреймворку Entity Framework Core, що дозволяє використовувати LINQ-запити для маніпуляції даних.

Вона описує реалізацію системи та взаємодію між її окремими вузлами. Для відображення схеми в даній роботі використовувалися наступні позначення: Компонент, Наданий інтефейс, Необхідний інтерфейс, Залежність, Підсистема.

Компонент - автономна одиниця системи, яка має виконувати певну поставлену задачу. Позначається прямокутником з двома виступами з лівого боку.

Наданий інтерфейс - позначається як лінія з невеликим колом на кінці. Вказує на те, що компонент надає іншим частинам системи певні послуги або операції.

Необхідний інтерфейс - зображується як півколо та вказує на те, що компонент потребує зовнішні ресурси від інших частин системи.

Залежність - показує виклик функцій між компонентами та зображується пунктирною стрілкою.

Підсистема - рамка, яка групує компоненти за місцем розгортання.

2.6 Структура серверної частини проєкту

Багаторівнева структура дозволяє ізолювати компоненти системи та спростити подальший етап тестування та майбутню підтримку програмного продукту.

Серверна частина розділена на чотири частини, взаємодія між якими представлена на діаграмі пакетів(рисунок 2.3).

Дана діаграма відображає логічну структуру і залежності пакетів системи, які далі групують для відображення великих програмних систем [10].

Основними елементами діаграми, які були використані при побудові діаграми проєкту були: Пакет, Відношення залежності, Стереотипи.

Пакети - об'єднання певних пов'язаних за функціональністю програмних елементів. В нашому випадку, вони відповідають за рішення .NET та Angular. Позначаються я прямокутники з виступом зверху.

Відношення залежності - відображається пунктирною стрілкою та вказує на те, що один компонент використовує дані з іншого компонента.

Стереотипи - показує певні властивості або особливості взаємодії між компонентами. В роботі використовуються `<<use>>`, що вказує на те, що пакет використовує функціональність іншого, викликаючи його методи чи маючи доступ до його інтерфейсів.

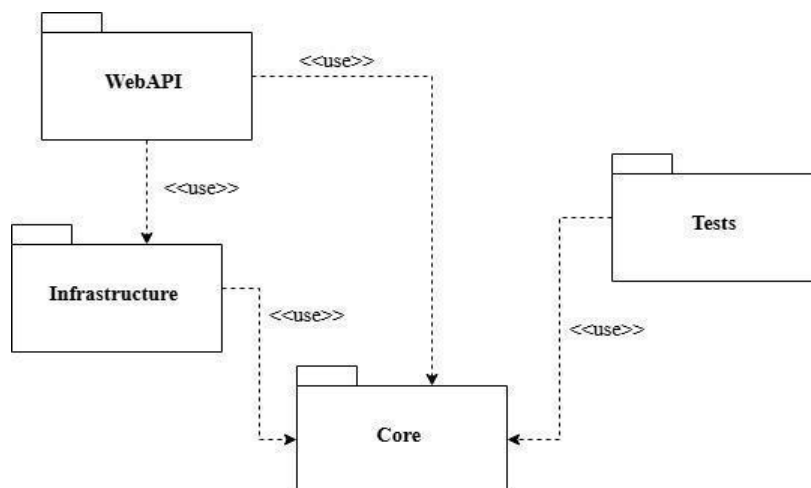


Рисунок 2.3 - Діаграма пакетів серверної частини

Складовими частинами сервера є:

1. AdvenTask.Core - рівень бізнес-сутностей.
2. AdvenTask.Infrastructure - рівень реалізації.
3. AdvenTask.WebAPI - рівень інтерфейсу.
4. AdvenTask.Tests - модуль тестування.

AdvenTask.Core - ядро системи. В ньому визначаються основні моделі даних системи: Quest, Artifact, Cosmetic, ApplicationUser, Difficulty, Tag. Quest, Artifact, Cosmetic та ApplicationUser. Ці об'єкти(entities) - описують структуру таблиць бази даних, а також застосовуються для формування DTO-об'єктів. Для переліку категорій класифікації завдань створенні списки Difficulty та Tag.

AdvenTask.Infrastructure - рівень, що містить опис структури бази даних та зв'язки між таблицями. Шар містить реалізацію логіки сервісів, включно з десеріалізацією ігрових ефектів артефактів із формату JSON та розрахунком фінального обсягу очок досвіду і кількість валюти для зарахування.

AdvenTask.WebAPI - вхідна точка для клієнтів. Містить контролери QuestController, ArtifactController, AccountController які маршрутизують вхідні запити користувачів та відповідають за автентифікацію через JWT-токени, ізоляцію даних користувачів.

AdvenTask.Tests - модуль для проведення автоматичного тестування логіки обрахування нагороди, застосування ефектів модифікаторів.

2.7 Архітектура клієнтської частини

Клієнтська частина проекту реалізована на базі фреймворку Angular та використання підходу компонентно-орієнтованого проектування. Інтерфейс вебзастосунка розбитий на невеликі, але незалежні одиниці(компоненти), кожна з яких керує своєю ділянкою екрана і містить в собі необхідні для роботи

компоненти [17]. Через взаємодію з Angular Router(authGuard) реалізовано перевірку прав доступу до внутрішніх сторінок.

Кожна функціональна область(сторінки магазину, квестів, користувача) має власний сервіс Angular (AuthenticationService, QuestService, ArtifactService, Cosmeticservice). Вони складають сервісний шар програмного комплексу та відповідає за взаємодію з сервером та реалізацію бізнес-логіки. Компоненти не знають про структуру запитів http, лише звертаються до методів сервісів, які використовують модуль HttpClient для асинхронного обміну даних. Таким чином, стає можливим застосовувати принцип DRY - один і той самий метод може викликатися з різних місць, уникаючи дублювання. До прикладу оновлювати профіль як у модулі управління завданнями для оновлення балансу, після отримання нагородження, так і в профілі користувача для ініціалізації даних.

Синхронізація між різними частинами інтерфейсу виконується через механізм Angular Signals. Дана технологія дозволяє відстежувати зміни ігрових метрик додатку(очі досвіду, монети та рівень користувача) та в автоматичному режимі оновлювати тільки ті компоненти, що безпосередньо прослуховують певний сигнал та залежать від змінених даних [18].

Цілісність та безпека сесій клієнтів реалізовано через комбінацію роботи jwtInterceptor, який перехоплює HTTP-запити користувачів та authGuard, що контролює доступ до маршрутів. Інтерцептор зчитує актуальний токен і якщо він наявний в системі, сесія користувача триває, якщо токен видалений - доступ до інформації блокується. authGuard ж перевіряє токен автентифікації. Якщо токен відсутній, або ж невалідний, то відбувається блокування до внутрішніх сторінок, тобто до сторінок квестів, облікового запису та магазину, а користувач перенаправляється на сторінку авторизації.

На діаграмі пакетів клієнтської частини (Рисунок 2.4) схематично відображено архітектурну взаємодію шару.

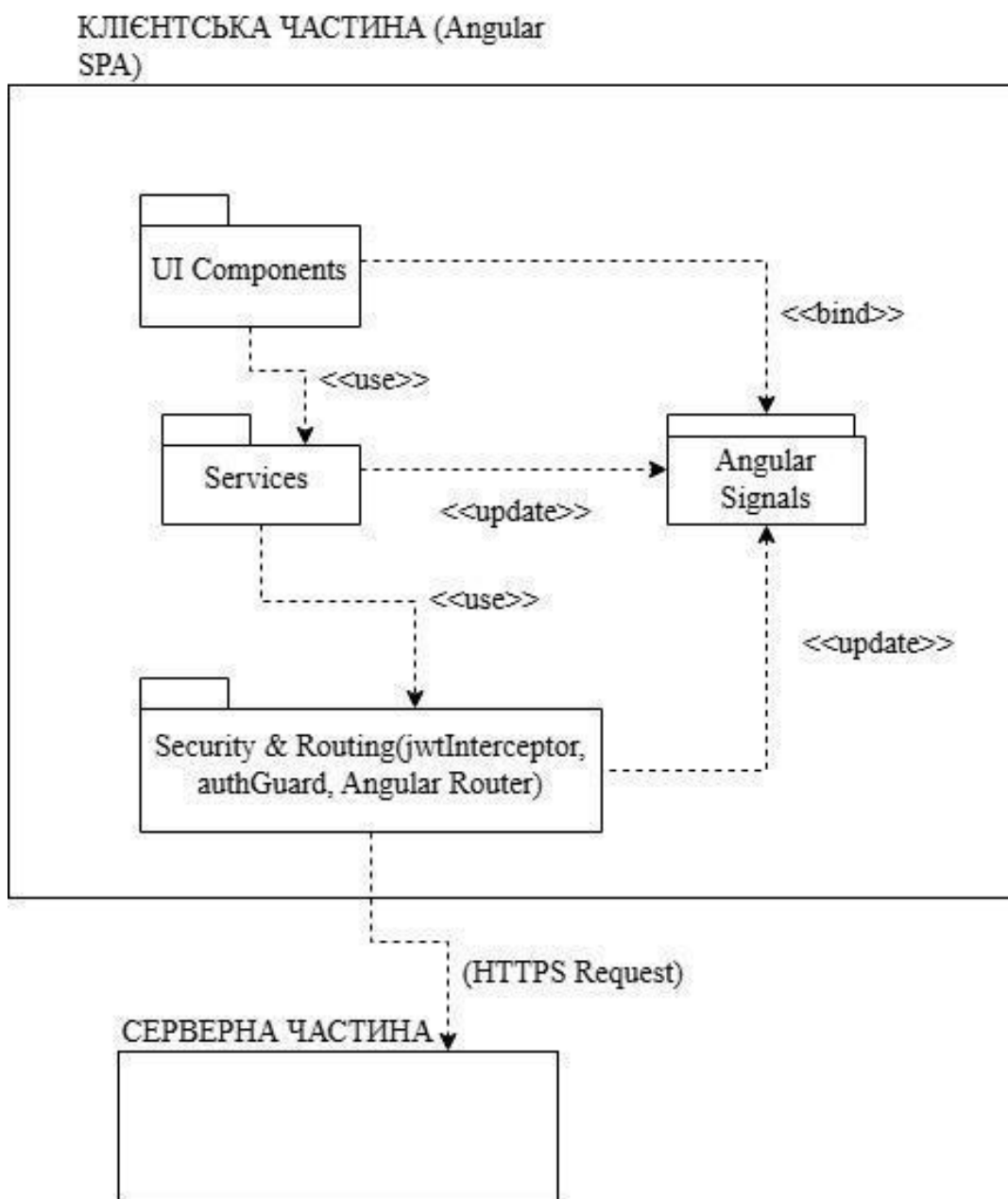


Рисунок 2.4 - Діаграма компонентів клієнтської частини

Додатково були додані два стереотипи:

<<update>> - позначення, що вказує на зміну внутрішнього стану об'єкта

<<bind>> - позначає механізм прив'язки, тобто компоненти інтерфейсу синхронізуються в автоматичному порядку з наявним станом даних.

2.8 Архітектура ігрової логіки вебзастосунку

Ігрова логіка системи “AdvenTask” базується на ігровому балансі та системі прогресії нагород, що сприяє залученню користувача. Центральний елемент розділу - алгоритм обробки завершених завдань. Процес нарахування винагороди побудований на наданні базових параметрів завдання(складність та тег). На виході, згідно наданих параметрів та застосованих на момент виконання артефактів, формується скоригований обсяг очок досвіду та віртуальної валюти.

Замість кодування кожної з властивостей предметів, логіка впливу структурованих у форматі JSON та зберігається в базі даних. Програмний шар застосунку виконує десеріалізацію таких об'єктів під час виконання програмного коду з текстового опису у функціональні алгоритми [24]. Алгоритм підтримує два типи впливу: мультиплікативний(збільшення отримання нагороди у відсотковому відношенні) та адаптивний(додавання фіксованої кількості вироблених ресурсів).

Алгоритм прогресії персонажа відслідковує накопичений обсяг очок досвіду та підвищує рівень профілю клієнта. Підняття рівню надає доступ до нових, потужних артефактів.

Ігрова логіка передбачає обмеження з боку клієнта для врівноваження балансу. Одночасно у користувача може бути застосовано чотири модифікаторів прогресу, відповідно закликає обирати між артефактами, щоб здобути максимальний прибуток. Крім того, в алгоритмі враховується фактор своєчасного виконання задач. У разі порушення часового обмеження, ефекти покращення від модифікаторів не враховуються.

Покроково процес розрахунку та нарахування ресурсів можна описати наступним чином:

- Користувач у графічному інтерфейсі позначає завдання як “Виконане”, що в свою чергу активує запит до серверної частини.
- Далі відбувається валідація термінів виконання. Система порівнює поточний час із встановленим дедлайном завдання.

- Якщо термін порушено - виконується процедура скасування дії активних модифікаторів.
- Якщо ж завдання виконане вчасно - алгоритм переходить до наступного кроку без втрати бонусів.
- Система звертається до бази даних для отримання списку артефактів, які наразі використовуються.
- Для кожного з екіпірованого артефакта відбувається десеріалізація текстового поля з властивостями об'єкта. Текст перетворюється на функціональний об'єкт, що містить поля з типом нагороди (очки досвіду або монети), операції (бонус до винагороди є статичним чи мультиплікативним) та її значення.
- На основі базової вартості завдань(що залежить від зазначеного рівня складності) відбувається обчислення за наступною формулою: Нагорода = База $\times$ Множник + Бонус.
- Фінальний результат записується до бази даних.
- Система порівнює поточну кількість очок та необхідну кількість для підняття рівня.
- Якщо досвіду достатньо - рівень піднімається, шкала досвіду обнуляється, з урахуванням залишку від підняття.
- Якщо досвіду недостатньо - рівень залишається незмінним і системи переходить до наступного кроку не виконуючи маніпуляції з даними.
- Після проведення розрахунків серверної частини додатку, активується механізм Angular Signals та відбувається оновлення інтерфейсу користувача без потреби перезавантаження сторінки.

Схематичне зображення алгоритму зображено на рисунку 2.5.

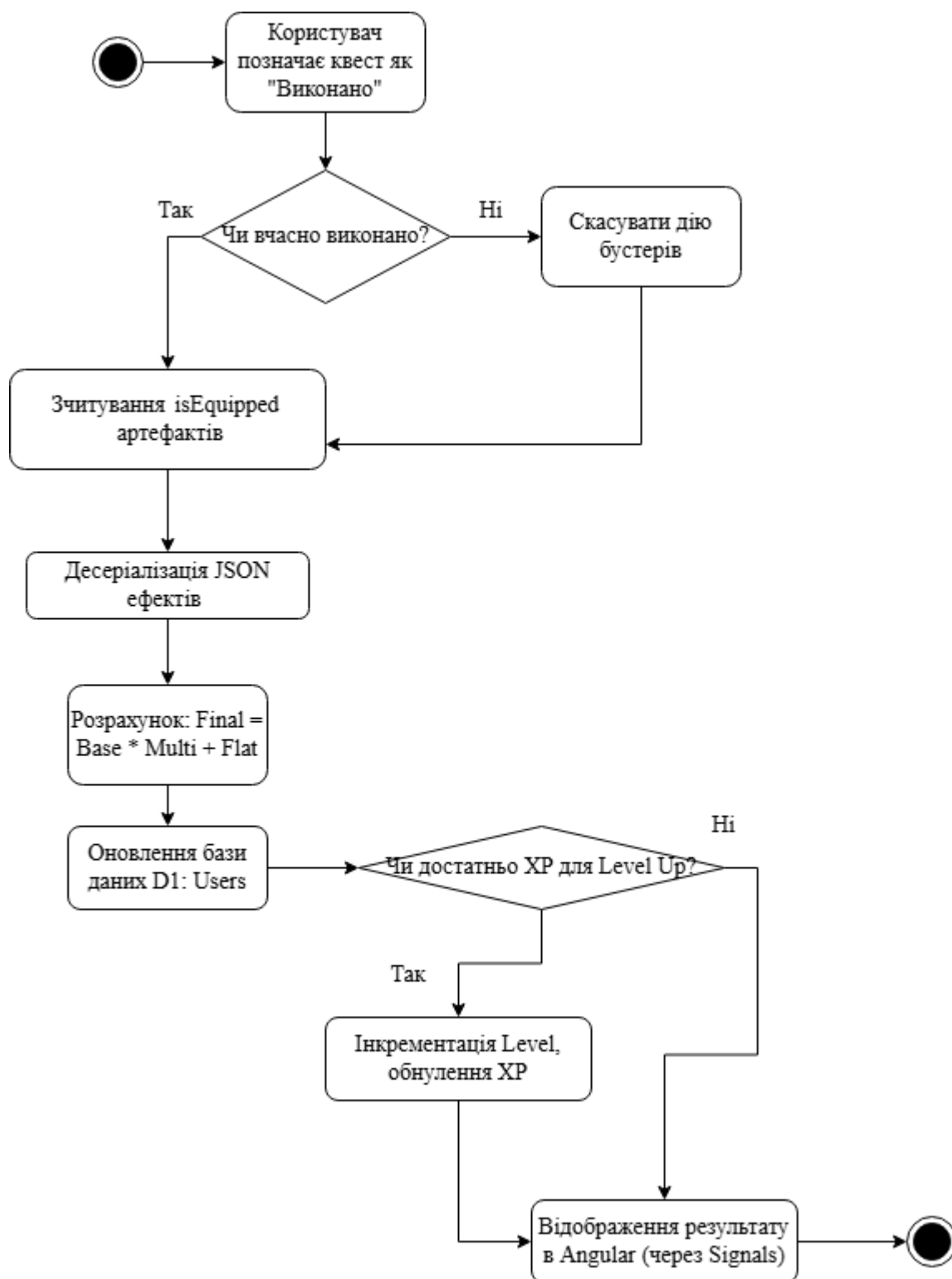


Рисунок 2.5 - Діаграма діяльності алгоритму розрахунку винагороди при виконанні завдання

Діаграма візуалізує цілісну роботу системи та є більш розширеним варіантом стандартних блок-схем. Серед позначень було використано: Початок/Кінець, Дії, Умови.

2.9 Архітектура бази даних вебзастосунку

Концептуальна схема реляційної бази даних представлено на рисунку 2.6.

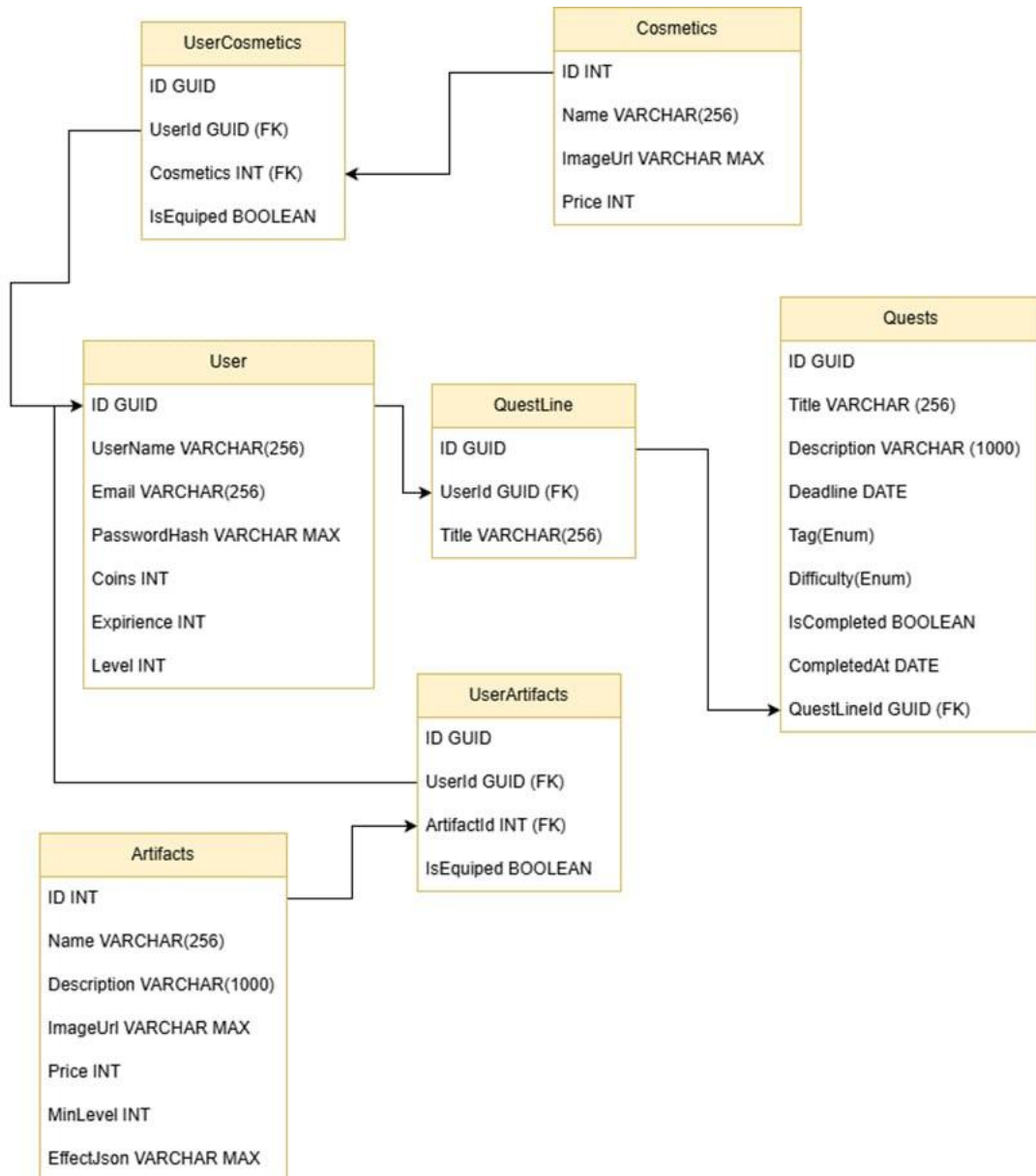


Рисунок 2.6 Концептуальна схема баз даних додатку

База даних складається з 7 таблиць.

Users - центральна таблиця, яка містить дані про зареєстрованих користувачів додатку. Також містить ігрові метрики(Рівень, кількість зарахованого досвіду та наявних монет).

Quests - таблиця з вмістом деталей про завдання. Через зв'язок багато-до-одного пов'язана з QuestLines.

QuestLines - таблиця-контейнер для групувань завдань у списки. При видаленні рядка QuestLines виконується каскадне видалення для всіх пов'язаних квестів. Має зв'язок багато-до-одного до таблиці Users.

Artifacts та Cosmetics - таблиці-довідники, що утримують перелік та інформацію про доступні для купівлі в магазині предметів, відповідно артефакти та аватари.

UserArtifacts та UserCosmetics - таблиці, що реалізують відношення багато-до-багатьох між користувачами та предметами. Зберігають факт володіння та стан використання.

База даних зведена до третьої нормальної форми (3NF). Кожне поле таблиці є атомарним, кожен неключовий стовпець залежить безпосередньо від первинного ключа. Таким чином, уникаються дублікати інформації та забезпечується швидке виконання запитів [25].

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У даному розділі описується практичне втілення описаної в попередньому розділі архітектури. Програмна реалізація включає налаштування бази даних, серверної частини на базі .NET Core та клієнтського інтерфейсу на Angular.

3.1 Програмна реалізація серверної частини

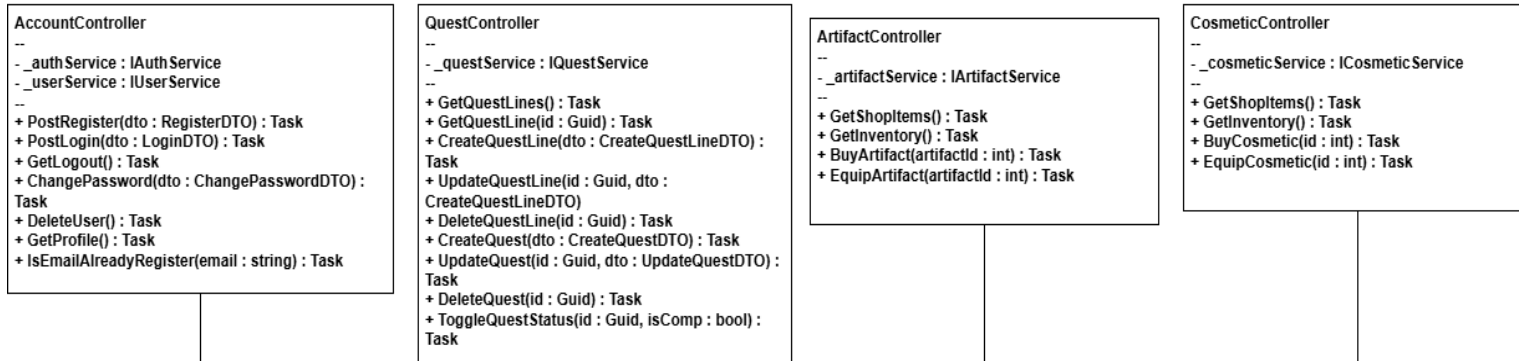
Програмний код сервера розподілений між трьома основними проєктами рішення: AdvenTask.Core, AdvenTask.Infrastructure та AdvenTask.WebAPI. Такий поділ дозволяє ізолювати доменну логіку від зовнішніх сервісів та фреймворків, що значно спрощує процес тестування та модернізації окремих вузлів системи. Загальна діаграма класів серверної частини додатка представлена на рисунку 3.1.

3.1.1 Реалізація об'єктної моделі та доступу до даних

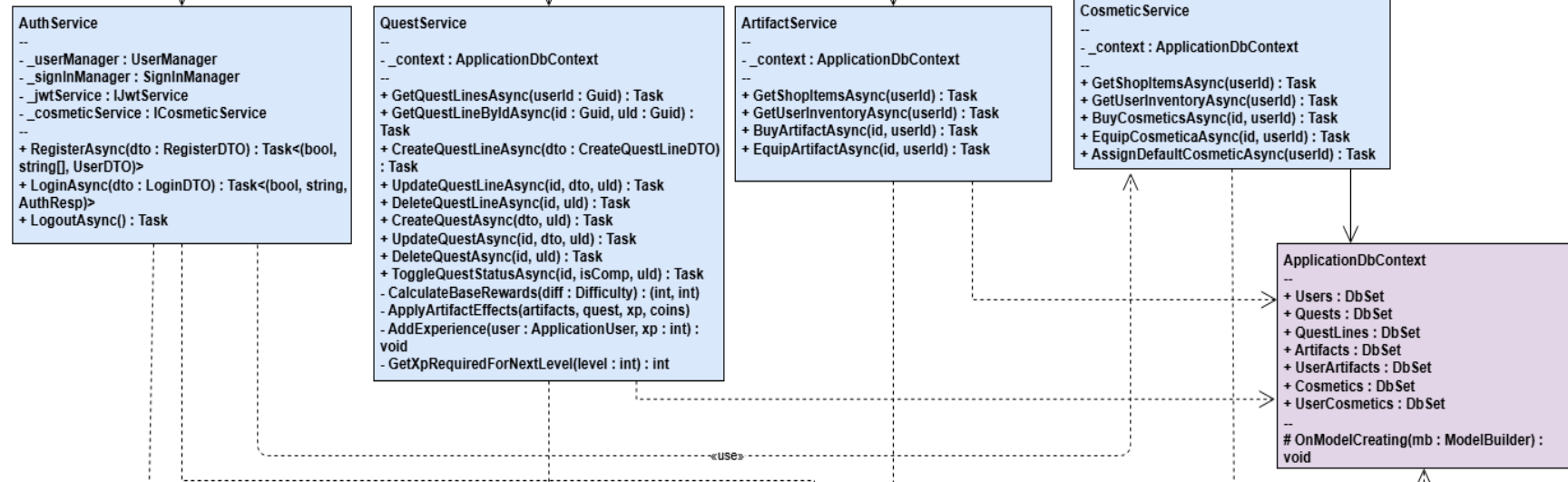
Практична реалізація системи планування задач першочергово передбачає формування об'єктної моделі для AdvenTask.Core, що формують структуру бази даних та логіку застосунку [26]. Цей клас містить тільки класи сутностей, згідно до запланованої архітектури додатка.

Для створення сутності користувача(ApplicationUser) використано стандарт ASP.NET Core Identity, що надає базові механізми автентифікації та авторизації [28]. Від нього отримується базовий набір полей(Email, PasswordHash, PhoneNumber), разом з ними і механізми безпеки, такі як автоматичне хешування паролів та захисту від підбірки паролів.

1. ШАР ПРЕДСТАВЛЕННЯ (REST API CONTROLLERS)



2. ШАР ІНФРАСТРУКТУРИ (BUSINESS SERVICES & DATA ACCESS)



3. ШАР ЯДРА (DOMAIN ENTITIES & INTERFACES)



Рисунок 3.1 - Діаграма класів серверної частини.

Тип Guid обрано для первинного ключа для збереженості унікальності та захисту. На відміну від цілочисельних ідентифікаторів, даний тип дозволяє забезпечити повну унікальність та неповторність ключа і ускладнює можливість перебору ідентифікаторів при несанкціонованому доступі. Для реалізації запланованих ігрових механік проєкту, клас ApplicationUser (Рисунок 3.2) доповнено полями Level, Experience, Coins. Також клас доповнений колекціями QuestLines, UserArtifacts, UserCosmetics. Дані колекції реалізують зв'язки з іншими функціональними модулями системи.

В проєкті використовується дворівнева модель завдань. В одному списку може міститися ціла низка завдань.

```
using AdvenTask.Core.Entities;
using Microsoft.AspNetCore.Identity;

namespace AdvenTask.Core.Identity
{
    public class ApplicationUser : IdentityUser<Guid>
    {
        public int Level { get; set; } = 1;
        19 references
        public int Experience { get; set; } = 0;
        24 references
        public int Coins { get; set; } = 0;

        0 references
        public ICollection<QuestLine> QuestLines { get; set; } = new List<QuestLine>();
        1 reference
        public ICollection<UserArtifact> UserArtifacts { get; set; } = new List<UserArtifact>();
        1 reference
        public ICollection<UserCosmetic> UserCosmetics { get; set; } = new List<UserCosmetic>();
    }
}
```

Рисунок 3.2 - Клас ApplicationUser

Quest - одиниця в QuestLine (Рисунок 3.3). Крім стандартних полів (назва, дата завершення, опису), клас використовує поля з фіксованим переліком (Enum) Difficulty (Easy, Medium, Hard) та Tag (Study, Sport, Work, Home, Social, Health), які потрібні в якості вхідних параметрів для логіки нарахування винагороди за наявності певних модифікаторів.

```

using System.ComponentModel.DataAnnotations;

namespace AdvenTask.Core.Entities
{
    24 references
    public class Artifact
    {
        [Key]
        24 references
        public int Id { get; set; }
        23 references
        public required string Name { get; set; }
        23 references
        public required string Description { get; set; }
        23 references
        public required string ImageUrl { get; set; }
        25 references
        public required int Price { get; set; }
        25 references
        public required int MinLevel { get; set; }
        23 references
        public required string EffectsJson { get; set; }
    }
}

```

Рисунок 3.3 - Клас Quest

QuestLine - контейнер завдань, що дозволяє розподілити завдання за контекстом, як відбувається в RPG-іграх. Структура класу представлена на рисунку 3.4.

```

using AdvenTask.Core.Identity;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace AdvenTask.Core.Entities
{
    public class QuestLine
    {
        [Key]
        public Guid Id { get; set; }

        [Required(ErrorMessage = "User id can't be blank")]
        [ForeignKey("User")]
        public Guid UserId { get; set; }

        [Required(ErrorMessage = "Title can't be blank")]
        13 references
        public required string Title { get; set; }

        4 references
        public ApplicationUser User { get; set; } = null!;

        4 references
        public ICollection<Quest> Quests { get; set; } = new List<Quest>();
    }
}

```

Рисунок 3.4 - Клас QuestLine

Сутність Artifact(Рисунок 3.5) виступає одним з елементів економіки ігрової механіки. Артефакт безпосередньо впливає на математичну модель нарахування віртуальних грошей. За те, який саме вплив буде здійснено впливає поле EffectsJson, що зберігає властивості модифікатора в текстовому форматі, що дозволяє легко додавати нові типи артефактів. MinLevel - поле, що обмежує доступ до високопродуктивних артефактів, доки користувач не підніме власний рівень досвіду.

```
using System.ComponentModel.DataAnnotations;

namespace AdvenTask.Core.Entities
{
    public class Artifact
    {
        [Key]
        public int Id { get; set; }
        public required string Name { get; set; }
        public required string Description { get; set; }
        public required string ImageUrl { get; set; }
        public required int Price { get; set; }
        public required int MinLevel { get; set; }
        public required string EffectsJson { get; set; }
    }
}
```

Рисунок 3.5 - Клас Artifact

Ще один клас, який є безпосереднім ігровим елементом - сутність Cosmetic(Рисунок 3.6). На відміну від артефактів, аватари мають суто декоративний характер і не впливають на математичну модель нагород.

```
using System.ComponentModel.DataAnnotations;

namespace AdvenTask.Core.Entities
{
    16 references
    public class Cosmetic
    {
        [Key]
        17 references
        public int Id { get; set; }
        15 references
        public required string Name { get; set; }
        17 references
        public required int Price { get; set; }
        15 references
        public required string ImageUrl { get; set; }
    }
}
```

Рисунок 3.6 - Клас Cosmetic

Зв'язок між користувачем та ігровими складовими реалізовано через проміжні сутності UserArtifact та UserCosmetic (Рисунок 3.7). Класи є втіленням відношення “багато-до-багатьох”. Ключова властивість сутностей - поле IsEquipped. Дане поле дозволяє увімкнути декілька артефактів одночасно та відображати застосований в даний момент аватар.

```
using System.ComponentModel.DataAnnotations;

namespace AdvenTask.Core.Entities
{
    public class UserArtifact
    {
        [Key]
        public Guid Id { get; set; }

        public Guid UserId { get; set; }

        public int ArtifactId { get; set; }

        public required Artifact Artifact { get; set; }
        10 references
        public bool IsEquipped { get; set; }
    }
}

using System.ComponentModel.DataAnnotations;

namespace AdvenTask.Core.Entities
{
    public class UserCosmetic
    {
        [Key]
        public Guid Id { get; set; }

        public Guid UserId { get; set; }

        public int CosmeticId { get; set; }

        public required Cosmetic Cosmetic { get; set; }

        public bool IsEquipped { get; set; }
    }
}
```

Рисунок 3.7 - Класи UserArtifact та UserCosmetic

На основі сформованих сутностей збудовано DTO-об'єкти. Вони виступають посередником між базою даних і застосунком. Всього у проєкті

створено дванадцять DTO-об'єктів, кожен з яких необхідний під свою задачу. Саме перетворення проходить через процес ручного мапінгу у шарі сервісів.

RegisterDTO - використовується для передачі даних між шарами архітектури для створення нового акаунту, LoginDTO ж призначений для передачі даних для ідентифікації користувача при вході в систему (Рисунок 3.8).

```
using System.ComponentModel.DataAnnotations;

namespace AdvenTask.Core.DTO.UserDTOs
{
    public class RegisterDTO
    {
        [Required(ErrorMessage = "Name can't be blank")]
        public string UserName { get; set; } = string.Empty;

        [Required(ErrorMessage = "Email can't be blank")]
        [EmailAddress(ErrorMessage = "Email should be in a proper email address format")]
        public string Email { get; set; } = string.Empty;

        [Required(ErrorMessage = "Password can't be blank")]
        3 references
        public string Password { get; set; } = string.Empty;

        [Required(ErrorMessage = "ConfirmPassword can't be blank")]
        [Compare("Password", ErrorMessage = "Password and confirm password do not match")]
        1 reference
        public string ConfirmPassword { get; set; } = string.Empty;
    }
}
```

```
using Microsoft.AspNetCore.Mvc;
using System.ComponentModel.DataAnnotations;

namespace AdvenTask.Core.DTO.UserDTOs
{
    public class LoginDTO
    {
        [Required(ErrorMessage = "Email can't be blank")]
        [EmailAddress(ErrorMessage = "Email should be in a proper email address format")]
        public string Email { get; set; } = string.Empty;

        [Required(ErrorMessage = "Password can't be blank")]
        public string Password { get; set; } = string.Empty;
    }
}
```

Рисунок 3.8 - DTO-об'єкти RegisterDTO, LoginDTO

AuthenticationResponse повертає JWT-токен та початкову статистичну інформацію користувача (Рисунок 3.9).

```

namespace AdvenTask.Core.DTO.UserDTOs
{
    public class AuthenticationResponse
    {
        public string? Token { get; set; } = string.Empty;
        public Guid UserId { get; set; }
        public string? UserName { get; set; } = string.Empty;
        public string? Email { get; set; } = string.Empty;
        public DateTime Expiration { get; set; }
        public int Level { get; set; }
        public int Experience { get; set; }
        public int ExperienceToNextLevel { get; set; }
        public int Coins { get; set; }
    }
}

```

Рисунок 3.9 DTO - об'єкт AuthenticationResponse

UserDTO(Рисунок 3.10) представляє спрощену модель профілю клієнта. Об'єкт використовується для оновлення статистичних даних. Не містить паролів та системних ідентифікаторів.

```

namespace AdvenTask.Core.DTO.UserDTOs
{
    public class UserDTO
    {
        public Guid Id { get; set; }
        public string? Name { get; set; }
        public string? Email { get; set; }
        public int Level { get; set; }
        public int Experience { get; set; }
        public int ExperienceToNextLevel { get; set; }
        public int Coins { get; set; }
    }
}

```

Рисунок 3.10 - DTO-об'єкт UserDTO

ChangePasswordDTO спеціалізований об'єкт для безпечної зміни пароля, що вимагає підтвердження поточної та нової комбінацій(Рисунок 3.11).

```

using System.ComponentModel.DataAnnotations;

namespace AdvenTask.Core.DTO.UserDTOs
{
    public class ChangePasswordDTO
    {
        [Required(ErrorMessage = "Current password is required")]
        public string CurrentPassword { get; set; } = string.Empty;

        [Required(ErrorMessage = "New password is required")]
        public string NewPassword { get; set; } = string.Empty;

        [Required(ErrorMessage = "Confirm new password is required")]
        [Compare("NewPassword", ErrorMessage = "The new password and confirmation password do not match.")]
        public string ConfirmNewPassword { get; set; } = string.Empty;
    }
}

```

Рисунок 3.11 - DTO-об'єкт ChangePasswordDTO

QuestDTO - повна модель завдань клієнтів. QuestLineDTO представляє групу завдань і містить список QuestDTO(Рисунок 3.12).

```

using AdvenTask.Core.Enums;
using System;
using System.Collections.Generic;
using System.Text;

namespace AdvenTask.Core.DTO.QuestDTOs
{
    public class QuestDTO
    {
        public Guid Id { get; set; }
        public Guid QuestLineId { get; set; }
        public required string Title { get; set; }
        public string? Description { get; set; }
        public DateTime? Deadline { get; set; }
        public Difficulty Difficulty { get; set; }
        public Tag? Tag { get; set; }
        public bool IsCompleted { get; set; }
        public DateTime? CompletedAt { get; set; }
    }
}

```

```

using System;
using System.Collections.Generic;
using System.Text;

namespace AdvenTask.Core.DTO.QuestDTOs
{
    public class QuestLineDTO
    {
        public Guid Id { get; set; }
        public required string Title { get; set; }
        public List<QuestDTO> Quests { get; set; } = new();
    }
}

```

Рисунок 3.12 - DTO-об'єкти QuestDTO та QuestLineDTO

Для оперування даними завдань було створено три об'єкта: CreateQuestLineDTO, CreateQuestDTO, UpdaeQuestDTO. Об'єкти представлено на рисунку 3.13.

```

using AdvenTask.Core.Enums;
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Text;

namespace AdvenTask.Core.DTO.QuestDTOs
{
    public class UpdateQuestDTO
    {
        [Required(ErrorMessage = "Title is required")]
        [MaxLength(200)]

        public string Title { get; set; } = string.Empty;

        [MaxLength(1000)]

        public string? Description { get; set; }

        public DateTime? Deadline { get; set; }

        [Required]

        public Difficulty Difficulty { get; set; }

        public Tag? Tag { get; set; }
    }
}

```

```

using AdvenTask.Core.Enums;
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Text;

namespace AdvenTask.Core.DTO.QuestDTOs
{
    public class CreateQuestDTO
    {
        [Required]

        public Guid QuestLineId { get; set; }

        [Required(ErrorMessage = "Title is required")]
        [MaxLength(200)]

        public string Title { get; set; } = string.Empty;

        [MaxLength(1000)]

        public string? Description { get; set; }

        1 reference
        public DateTime? Deadline { get; set; }

        [Required]
        2 references
        public Difficulty Difficulty { get; set; }

        1 reference
        public Tag? Tag { get; set; }
    }
}

```

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Text;

namespace AdvenTask.Core.DTO.QuestDTOs
{
    public class CreateQuestLineDTO
    {
        [Required(ErrorMessage = "Title is required")]
        2 references
        public string Title { get; set; } = string.Empty;
    }
}

```

Рисунок 3.13 - DTO-об'єкти CreateQuestLineDTO, CreateQuestDTO, UpdaeQuestDTO

CreateQuestLineDTO - мінімалістичний об'єкт, що використовується для створення нових гілок завдань, містить лише назву. Об'єкт CreateQuestDTO використовується для додавання нового завдання. На відміну від QuestDTO, цей об'єкт не містить поля ідентифікатора та поля статусу екіпірування. UpdaeQuestDTO дозволяє редагувати вже створенні задачі, але перелік полів, які доступні для змінення є обмеженими.

ArtifactDTO та CosmeticDTO містять інформацію про ігровий предмет. DTO-об'єкти представлені на рисунку 3.14.

```
namespace AdvenTask.Core.DTO.ShopDTOs
{
    8 references
    public class ArtifactDTO
    {
        2 references
        public int Id { get; set; }
        2 references
        public required string Name { get; set; }
        2 references
        public required string Description { get; set; }
        2 references
        public required string ImageUrl { get; set; }
        2 references
        public int Price { get; set; }
        2 references
        public int MinLevel { get; set; }
        2 references
        public bool IsOwned { get; set; }
        2 references
        public bool IsEquipped { get; set; }
    }
}
```

```
namespace AdvenTask.Core.DTO.ShopDTOs
{
    public class CosmeticDTO
    {
        public int Id { get; set; }
        public required string Name { get; set; }
        public required string ImageUrl { get; set; }
        public int Price { get; set; }
        public bool IsOwned { get; set; }
        public bool IsEquipped { get; set; }
    }
}
```

Рисунок 3.14 - DTO-об'єкти ArtifactDTO та CosmeticDTO

Крім статичних полів, об'єкти мають два динамічні поля - IsOwned(визначає чи придбаний цей товар), IsEquipped(чи активований він в даний момент).

Взаємодія реляційної бази даних відбувається завдяки ORM Entity Framework Core. Доступ до даних організовано через клас ApplicationDbContext, успадкований від IdentityDbContext [13].

3.1.2 Шар сервісів та реалізації бізнес-логіки зарахування нагороди

Шар сервісів є основним інструментом реалізації сценарії використання додатку. Завдяки використанню механізму Dependency Injection (DI), сервіси отримують доступ до контексту бази даних та взаємодіють між собою, зберігаючи низьку зв'язність компонентів [11].

Група сервісів AuthService, JwtService, UserService є сервісами управління обліковими записами. AuthService виконує реєстрацію нових користувачів та перевірку даних при вході в вже створені облікові записи, також відповідає за вихід з системи. При реєстрації, сервіс ініціює виклик до сервісу CosmeticService, таким чином призначаючи новому користувачеві стандартний аватар. JwtService розроблений для створення зашифрованих токенів сесій. Сервіс використовує алгоритм HmacSha256 та формує набір даних (ID користувача, Email, Ім'я) для ідентифікації клієнта [14]. UserService слугує допоміжним сервісом для вирішень задач оновлення актуальної інформації про обліковий запис, зміни пароля чи видалення профілю. За допомогою метода GetUserProfileAsync розраховує динамічне поле ExperienceToNextLevel.

QuestService реалізує управління статусом виконання задач, цикл CRUD для ліній квестів та окремих завдань. Також сервіс містить логіку нарахування валюти та досвіду. Якщо досвід перевищує ліміт - відбувається інкрементація рівня користувача. Відповідно до вказаної при створенні чи редагуванні складності, перевіряє дедлайн та застосовує формули з активних модифікаторів. Обчислення винагороди відбувається через приватний метод ApplyArtifactEffects, що десеріалізує JSON-опис ефектів активних артефактів та

застосовує мультиплікативні або адитивні модифікатори до базових значень очок досвіду та валюти [24].

ArtifactService та CosmeticService є сервісами, що керують віртуальними активами. В сервісі ArtifactService зосереджено відображення та купівлю модифікаторів. Обмежує кількість застосованих одночасн модифікаторів до чотирьох. Ключовий метод для розрахування нагороди - ApplyArtifactEffects, який використовує десеріалізацію JSON. Метод покупки BuyArtifactAsync реалізує транзакційну перевірку двох умов: достатності балансу (Coins) та відповідності рівня користувача вимогам предмета (MinLevel).

CosmeticService реалізує подібну логіку, але застосовується до аватарів. Може бути активований тільки один аватар, тому при встановленні нового аватара, у попереднього сервіс змінює значення поля IsEquipped на false.

3.1.3 Шар представлення. Реалізація REST API контролерів

Рівень представлення у проєкті реалізовано у AdvenTask.WebAPI через відповідні контроллери. Цей рівень має маршрутизовувати вхідні HTTPS запити, виконувати валідацію DTO-об'єктів та повертати оброблений результат.

AccountController виступає вхідною точкою для логіна реєстрації та логіну. Контроллер має наступні ключові методи:

- PostRegister - реєстрація нового користувача, включаючи ініціалізацію стартових метрик.
- PostLogin - автентифікація користувача, повернення метрик облікового запису.
- GetProfile - метод для синхронізації при завантаженні сторінок.
- ChangePassword - зміна пароля профілю користувача.
- DeleteUser - видалення облікового запису клієнта.

QuestController є елементом для роботи з планами завдань. Включає наступні методи:

- GetQuestLines та GetQuestLine - відповідають за отримання структурованих списків завдань та самих задач всередині груп.
- CreateQuestLine, UpdateQuestLine, DeleteQuestLine - методи для операцій з створення, видалення або оновлення груп задач.
- CreateQuest, UpdateQuest, DeleteQuest - аналогічно до попередніх, проте оперує вже самим завданнями.
- ToggleQuestStatus - метод, що ініціалізує цикл ігрового елементу. Метод використовує PATCH для часткового оновлення об'єкта. Замість повного переписування, такий підхід дозволяє змінити тільки поле isCompleted для запуску алгоритму нарахування нагороди.

ArtifactController відповідає за ігровий магазин модифікаторів. Ключові методи контроллера:

- GetShopItems - повертає список всіх існуючих в додатку артефактів. Відповідно до значення IsOwned, предмет буде доступний або до екіпірування або до купівлі.
- GetInventory - повертає список предметів, які було придбано. Застосовується при завантаженні сторінки профілю, демонструючи наявні предмети.
- BuyArtifact - точка ініціації транзакції з придбання товару. Включає і перевірку балансу з рівнем.
- EquipArtifact - ініціація активації артефактів.

CosmeticController

- GetShopItems GetInventory - аналогічно до попередньо контроллера застосовується по відношенню до аватарів.
- BuyCosmetic - точка ініціації придбання нового аватара.
- EquipCosmetic - ініціація зміни активованого аватара на новий.

3.1.4 Конфігурація серверного середовища, впровадження залежностей

Конфігурація серверної частини та обробка запитів зосереджена у файлі Program.cs. Усі бізнес-сервіси впроваджуються через інтерфейси. Залежно від закладеного призначення, налаштовані відповідні життєві цикли - Scoped, Transient.

Scoped - створюється єдиний екземпляр сервісу для кожного вхідного запиту від користувача. Такий підхід є оптимальним рішенням для сервісів, що взаємодіють з базою даних. Цей життєвий цикл мають AuthService, QuestService, ArtifactService, CosmeticService, UserService.

Transient - створюється екземпляр при кожному виклику. Такий цикл має JwtService, оскільки він має специфічну задачу та не потребує спільного доступу до даних в БД.

В цьому файлі також визначена політика паролів, яка не дозволяє встановлювати паролі менше 5 символів та зобов'язує використовувати цифри, літери великого та малого реєстру.

В програмі використовується механізм JWT Bearer Authentication [14]. Сервер конфігуровано на досить строгу перевірку вхідних токенів через об'єкт Token Validation Parameters. Об'єкт має властивості Validate Issuer Signing Key, що підтверджує криптографічний підпис за допомогою секретного ключа (Jwt:Key) для збереження цілісності даних, validateIssuer та ValidateAudience виконують верифікацію видавця, щоб унеможливити використання токенів з інших середовищ, а ValidateLifetime відповідає за контроль терміну дії токена, щоб запобігти використанню застарілих токенів для несанкціонованої експлуатації старих сесій [29].

Оскільки клієнтська сторона застосунк знаходиться на окремому сервері, для безпечної взаємодії між ними налаштовано політику CORS(Cross-Origin Resource Sharing). Ця технологія дозволяє браузеру надавати вебзастосунку доступ до ресурсів іншого домену, обходячи обмеження політики одного

джерела (Same-Origin Policy) [30]. У межах розробленої системи конфігурація CORS у файлі Program.cs дозволено обробку необхідних HTTP-запитів та заголовків, що забезпечує стабільну роботу Angular-клієнта з REST API.

3.1.5 Модульне тестування серверної логіки

У якості основного фреймворку для реалізації модульного тестування було обрано xUnit [22]. Тести розташовані у проєкті AdvenTask.Tests та застосовуються два підходи: In-Memory Database та Mocking.

Перший підхід використано для тестів, що мають потребу взаємодіяти з базою даних. Він потребує провайдер до БД, яка функціонує тільки в межах оперативної пам'яті пристрою під час виконання самих тестів. Таким чином, виконується правдивий тест, без впливу на реальні таблиці даних [31].

Мокінг - техніка, яка замість реальних об'єктів використовує імітації, які вже мають певну визначену поведінку [32].

Тестове покриття охоплює критичні сценарії системи.

1. Перевірка ігрової економіки та її обмежень (ArtifactServiceTests, CosmeticServiceTests). Ці тести перевіряють валідацію транзакцій у магазині. Перевіряють списання валюти (BuyArtifact_EnoughCoinsAndLevel_Success), відмову при спробі придбання недоступних товарів через нестачу грошей чи досвіду (BuyArtifact_NotEnoughCoins_ReturnsError та BuyArtifact_Level TooLow_ReturnsError). Також реалізовано тест, що перевіряє ліміт на одночасне використання предметів та аватарів (EquipArtifact_LimitReached_ReturnsError).

2. Тестування механік нарахування винагород (QuestServiceTests). Тести перевіряють алгоритм застосування множників при розрахунку винагороди з використанням артефактів, коли система зчитує властивості модифікатора з JSON-об'єкта. Також перевіряється, чи переноситься надлишок досвіду до нового рівня.

3. Тестування реєстрації(AuthServiceTests). Тестується вхід (Login_ValidCredentials_ReturnsToken та Login_UserNotFound_ReturnsError) та реєстрація нового користувача. Також, перевіряється ініціалізація стартового косметичного аватара (Register_ValidData_CreatesUser_And_AssignsCosmetic).

3.2 Програмна реалізація клієнтської частини

Оскільки архітектурно клієнтська сторона додатку побудована слідуючи принципу Feature-based structure, кожен функціональний модуль є відокремленим один від одного [17].

Всі функціональні файли знаходяться у папці src/app/ і має наступний зміст:

core/ - вмістилище системних елементів. В заданій теці знаходяться сервіси взаємодії з API, моделі даних, Interseptors, Guards та глобальні компоненти сторінок.

features/ - тека з основним робочим простором, що розділений на зони за функціоналом. До цих зон відносяться:

- auth/ - сторінка реєстрації та входу
- quests/ - сторінка з менеджментом завдань
- shop/ - сторінка маркетплейсу з поділом на артефакти та косметику
- character/ - сторінка профілю користувача з його інвентарем
- settings/ - сторінка для зміни паролю та видалення акаунту

Кожен незалежний компонент має складові з *.ts(функціональна логіка та взаємодія з сервіса), *.html(розмітка інтерфейсу), *.scss(стилі дизайну).

Для здійснення переходів між зонами використовується файл app.routers.ts, в якому прописаний функціонал сигналів [19].

3.2.1 Управління станом через Angular Signals

Angular Signals дозволяють реалізувати гранулярне оновлення DOM-дерева лише при зміні конкретних значень, уникаючи зайвих обчислень [18]. Сигнали в системі «AdvenTask» класифіковані за трьома групами:

Загалом у проєкті можна поділити на три групи за функціональністю.

Глобальні сигнали для сервісів. Зберігаються в сервісі `AuthenticationService.ts`. `currentUser` - зберігає об'єкт облікового поточного користувача, що пройшов авторизацію, включаючи ігрові метрики. При зміні метрик профілю, викликається метод оновлення облікового запису і сигнал отримує дані від API та оновлюється даний сигнал. Відповідно, ті компоненти, що слухають цей сигнал оновлюються згідно до отриманої інформації. `isLoggedInSignal` - сигнал, що керує прапорцями доступу та видимістю елементів навігації.

Сигнали системних сервісів. Для реалізації декларативного інтерфейсу з користувачем використовується сигнал у сервісі `DialogService.ts` `dialogState` - сигнал, що містить об'єкт конфігурації модального вікна. Таким чином, системні сповіщення можна викликати з будь-якого місця програми, не створюючи локальні об'єкти.

Локальні сигнали компонентів. Дані сигнали використовуються безпосередні всередині певних компонентів для здійснення локальних задач. `showModal` - сигнал для ініціалізації рендерингу певного компонента, або ж його видалення з DOM-дерева. `selectedQuest` та `selectedLine` - сигнали, що зберігають об'єкт, який було обрано для редагування(в разі нульового значення, то з'являється форма для створення). `questLines` - зберігає масив папок завдань користувача. `selectedQuestLine` - зберігає інформацію про відкриту поточну гілку завдань. `currentUserCoins` та `currentUserLevel` - локальні сигнали, що копіюють стан при ініціалізації магазину. Вони динамічно допомагають відключати або включати кнопки купівлі в магазині. `activeQuests` та `completeQuests` - обчислювальні сигнали, що розділяють список завдань на активні та закриті.

3.2.2 Шар системної логіки та інфраструктури клієнтської частини додатку

Фундаментальна інфраструктура клієнтської частини застосунку міститься в папці `core`. Всі елементи в даній теці ініціалізуються один раз при запуску програми й використовуються глобально в усій частині системи.

Моделі даних та списки типізації (`core/models`). Каталог містить TypeScript-інтерфейси (`User.ts`, `Artifact.ts`, `Quest.ts`) та переліки (`enum.ts`). Загалом, дані об'єкти є точною копією моделей DTO, що знаходяться в серверній частині. Це забезпечує узгодженість даних між двома частинами програми - клієнтською та серверною [17].

Мережева взаємодія та сервіси (`core/services`). Сервіси (`QuestService`, `ArtifactService`, `CosmeticService`) використовують вбудований модуль `HttpClient` для реалізації методів доступу до REST API. `AuthenticationService` керує процесом входу і дозволяє компонентам ініціювати оновлення даних користувача через метод `refreshProfile()`, таким чином відбувається оновлення сигналу `currentUser` і миттєве оновлення ігрових метрик на сторінці додатку. `DialogService` є допоміжним для забезпечення асинхронного управління модальними вікнами.

Механізми безпеки та автоматизації (`core/interceptors` `core/guards`). `jwt-interceptor.ts` перехоплює кожен вхідний мережевий запит до сервера, зчитує актуальний токен JWT з локального сховища браузера та додає його до заголовка авторизації [19]. В той же час, `auth-guard.ts` забезпечує безпеку маршрутів додатку, перевіряючи статус авторизації поточного користувача перед тим, як завантажити сторінку. Якщо сесія недійсна - користувач перенаправляється на сторінку входу в додаток [19].

Візуальні компоненти додатка (`core/components`). У папці зосереджений загальний каркас застосунку. Навігаційна панель (`Navbar`) виступає ключовим елементом для перенаправлення користувачів до сторінок додатку. В реальному часі панель відображає ігрові метрики користувача. Компоненти `Model` та

SystemDialog використовуються як універсальні контейнери для відображення діалогових вікон.

Допоміжні утиліти(core/utils). enum-utils.ts надає функції для перетворення переліків у масиви, що стають придатними для відображення випадючих списків у формах.

3.2.3 Функціональні модулі

Директорія features слугує простором для реалізованих користувацьких інтерфейсів, кожен з яких є окремим компонентом.

Модуль автентифікації (auth) забезпечує вхід користувача в систему та реєстрацію нових акаунтів. Його компоненти Login та Register побудовані на базі механізму реактивних форм Angular (ReactiveFormsModule), тому можуть проводити складну валідацію даних на стороні клієнта (наприклад, перевірка коректності email або збігу паролів) ще до відправки запиту на сервер [19]. У разі виникнення помилок на стороні API, модуль динамічно обробляє масив помилок та візуалізує їх у вигляді повідомлень, стилізованих під ретро-естетику за допомогою бібліотеки NES.css[20].

Модуль управління завданнями(quests) є частиною функціоналу планувальник. Компонент QuestsLayout організовує простір як панель списків ліворуч(QuestLineList), з переліком завдань в обраній лінії завдань праворуч(QuestList). Через метод toggleStatus, коли змінюється статус чекбоксів завдань, ініціюється запит оновлення БД додатка.

Модуль магазину(shop) розділений на дві секції - персонажів та артефактів. Компоненти ShopArtifact та ShopCosmetics виконують динамічну перевірку умов покупок, відповідно відображають можливість придбання через візуал. Якщо не є можливим купити якийсь товар, кнопка буде сірою і бездіяльною, доки не будуть виконані умови.

Модуль персонажа (character) забезпечує візуалізацію досягнутого прогресу. Він інтегрує дані про екіпіровані артефакти та обраний аватар. Компонент відслідковує зміни в сигналі currentUser та оновлює прогрес-бари досвіду та лічильники валюти без перезавантаження сторінки.

3.2.4 Візуальне оформлення

Візуальне оформлення проєкта виконане в стилі 8-бітних ігор. Для цього було використано CSS-фреймворк NES.css[20]. Фреймворк надає набір готових стилізованих компонентів, що імітують графіку з високою роздільністю .

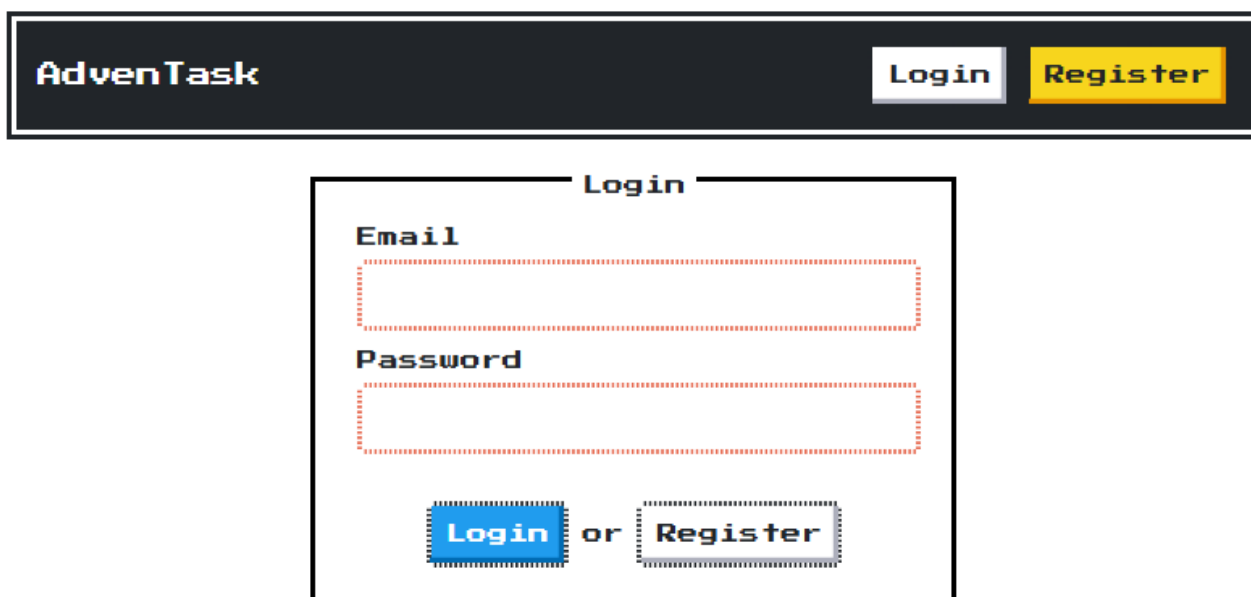
Основний шрифт застосунку - "Press Start 2P", підключений через Google Fonts. Для підвищення рівня занурення в ігровий процес у глобальному файлі стилів styles.scss реалізовано кастомізацію курсору: стандартний системний вказівник замінено на піксельне зображення руки, що змінює свій вигляд при наведенні на інтерактивні елементи.

Ще одним аспектом візуального оформлення є кольорове маркування пріоритетів та складності завдань. У компоненті QuestList реалізовано логіку динамічного призначення CSS-класів залежно від рівня складності завдання. Відповідно, квести категорії «Easy» отримують зелене забарвлення меж, «Medium» — жовте, а найбільш трудомісткі завдання «Hard» виділяються яскраво-червоним кольором.

Крім того, для збереження чіткості ігрових асетів (аватарів героїв та артефактів) у стилях застосовано властивість image-rendering: pixelated, яка блокує алгоритми згладжування браузерів, зберігаючи різкість пікселів при масштабуванні графіки [34].

4. РОБОТА КОРИСТУВАЧА З ПРОГРАМОЮ

Взаємодія користувача з програмною системою починається з етапу автентифікації. Для доступу до функціоналу застосунку необхідно пройти процедуру входу або створити новий обліковий запис. Система забезпечує валідацію введених даних клієнтом, надаючи користувачеві зворотний зв'язок у разі помилок. Зовнішній вигляд сторінки авторизації наведено на рисунку 4.1.



The image shows a web interface for 'AdvenTask'. At the top, there is a dark header bar with the 'AdvenTask' logo on the left and two buttons, 'Login' and 'Register', on the right. Below the header, there is a white box titled 'Login'. Inside this box, there are two input fields: 'Email' and 'Password', each with a red dashed border. Below these fields, there are two buttons: 'Login' (blue) and 'Register' (blue), separated by the word 'or'.

Рисунок 4.1 — Інтерфейс сторінки авторизації користувача

Головним робочим простором застосунку є розділ управління завданнями. Інтерфейс цього розділу розділено на дві основні зони. Ліворуч розташована бічна навігаційна панель, де користувач може створювати та керувати «Лініями квестів» (Quest Lines), задаючи категорію завдання за певними сферами життя, наприклад, «Навчання», «Робота» чи «Спорт». При виборі конкретної лінійки в основному вікні праворуч відображається перелік завдань, які розділені на активні та виконані.

Для додавання нового завдання користувач натискає відповідну кнопку, що викликає модальне вікно. У ньому вказується назва завдання, детальний опис, вибирається рівень складності, опціональний тематичний тег та кінцевий термін

виконання (дедлайн). З метою покращення візуального сприйняття кожне завдання в загальному списку має кольорову індикацію рамки залежно від його складності. Для завершення квесту користувачеві достатньо натиснути на чекбокс, після чого завдання перекреслюється і переноситься до списку виконаних, а в навігаційній панелі миттєво оновлюється баланс монет та досвіду. Процес управління завданнями зображено на рисунку 4.2.

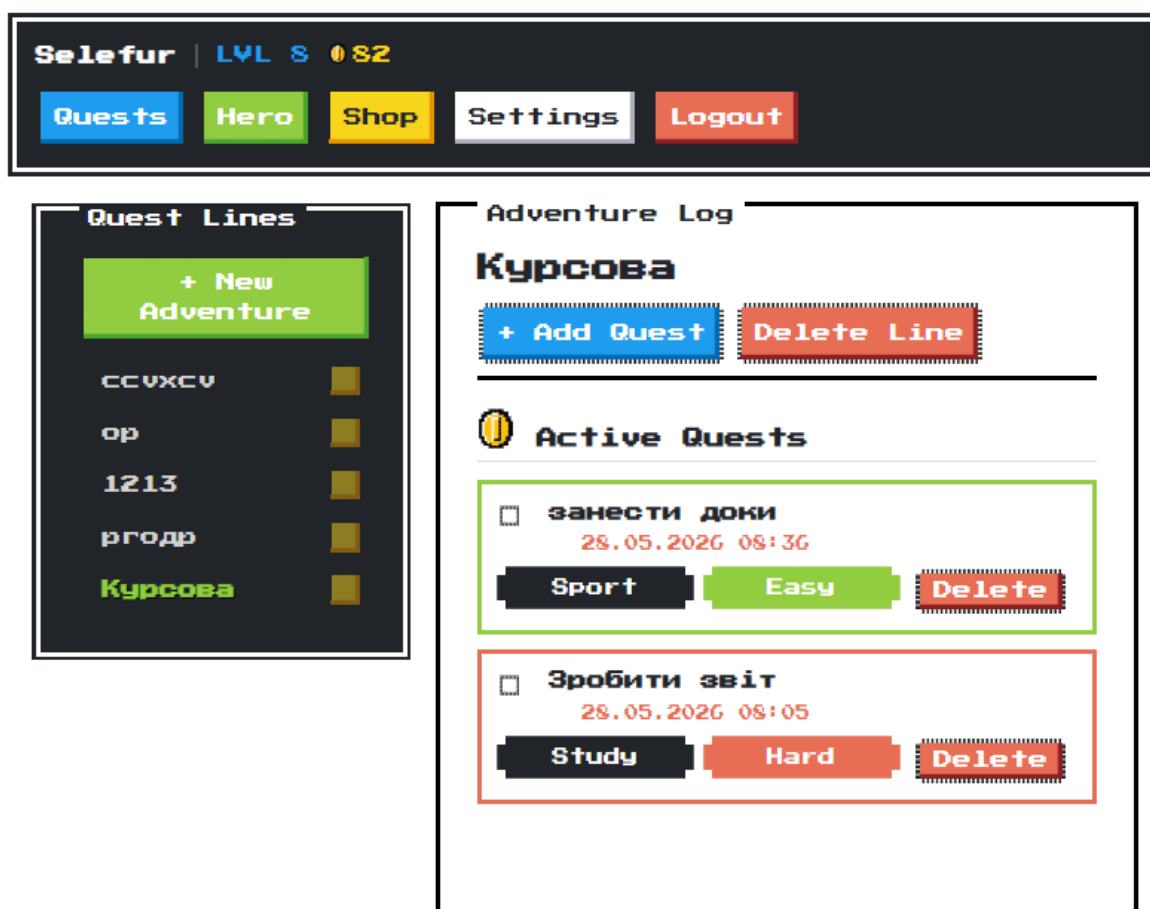


Рисунок 4.2 — Робочий простір управління завданнями (квестами)

Для відстеження ігрового прогресу та управління отриманими нагородами передбачено розділ профілю персонажа (Hero). Верхня частина сторінки містить інформаційну картку героя, де відображається його ім'я, візуальний аватар, поточний рівень та динамічна шкала накопиченого досвіду. Завдяки цій шкалі користувач може наочно бачити, скільки зусиль залишилося до досягнення наступного рівня.

Нижня частина сторінки відведена під інвентар артефактів. Усі придбані в магазині предмети відображаються у вигляді сітки слотів. Користувач має можливість клікнути на наявний артефакт, щоб екіпірувати його; при цьому на іконці предмета з'являється спеціальна позначка, а сам слот підсвічується. Оскільки система накладає обмеження на кількість одночасно екіпірованих предметів (не більше чотирьох), інтерфейс автоматично блокує можливість вибору нових артефактів після досягнення цього ліміту, відображаючи відповідне попередження. Інтерфейс профілю персонажа наведено на рисунку 4.3.

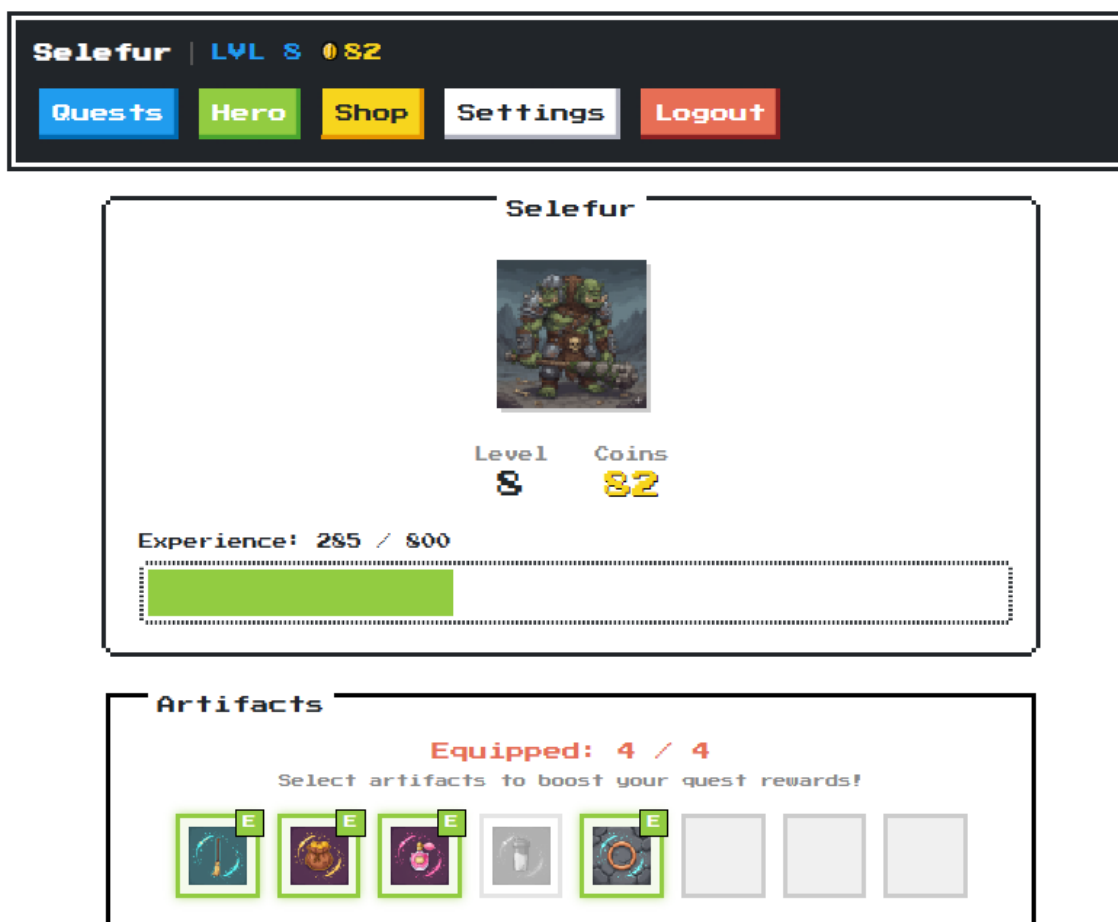


Рисунок 4.3 — Відображення профілю персонажа та інвентарю артефактів

Розділ віртуального магазину (Marketplace) надає користувачеві можливість конвертувати зароблені під час виконання завдань монети у цифрові активи. Магазин розділено на дві окремі вкладки: «Артефакти» (Artifacts) для

купівлі предметів, що дають математичні бонуси до нагород, та «Косметика» (Cosmetics) для придбання нових аватарів. Зовнішній вигляд віртуального магазину продемонстровано на рисунку 4.4.

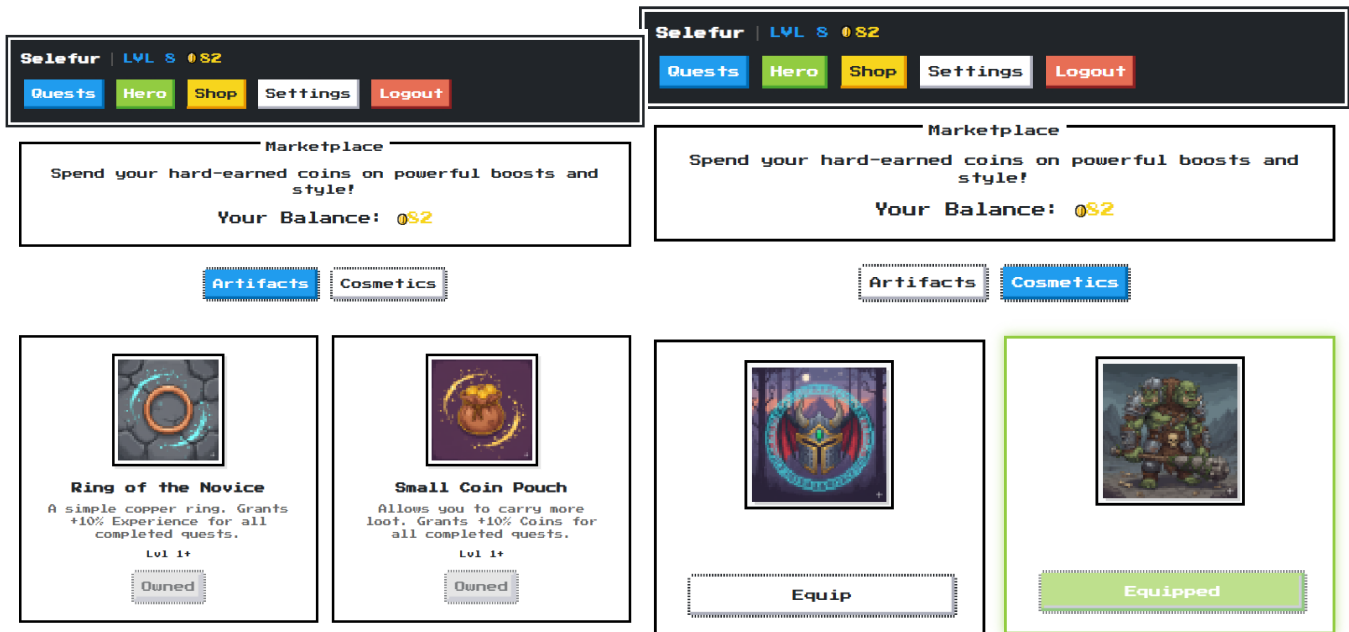


Рисунок 4.4 — Інтерфейс віртуального магазину системи

Кожен товар у магазині представлено у вигляді окремої картки, що містить графічне зображення, опис впливу на ігровий баланс, вартість у монетах та вимогу до мінімального рівня героя. Логіка інтерфейсу побудована таким чином, що кнопка покупки є інтерактивною лише за умови виконання всіх критеріїв: наявності достатньої кількості монет, відповідності рівню персонажа та відсутності даного предмета в інвентарі. У випадку успішного придбання баланс монет миттєво оновлюється без необхідності перезавантаження сторінки, а товар стає доступним для використання у профілі героя.

ВИСНОВКИ

За час виконання завдання дипломної роботи, було отримано досвід розробки сучасного програмного забезпечення на базі клієнт-серверної архітектури. Проведено аналіз предметної області та існуючих програмних рішень, що дозволило виявити основні недоліки класичних планувальників завдань та обґрунтувати необхідність створення гейміфікованої системи. На основі проведеного дослідження було спроектовано оптимізовану архітектуру реляційної бази даних та розроблено загальну структуру вебзастосунку «AdvenTask».

Практична реалізація серверної частини була успішно виконана на базі платформи .NET 10 з використанням фреймворку ASP.NET Core та технології об'єктно-реляційного відображення Entity Framework Core. Під час розробки бекенду створено гнучку архітектуру REST API, налаштовано безпечну систему автентифікації на базі JWT-токенів та успішно імплементовано бізнес-логіку гейміфікації. Розроблений серверний алгоритм забезпечує динамічне нарахування очок досвіду та віртуальної валюти, автоматичний розрахунок підвищення рівня персонажа, а також зчитування і застосування математичних модифікаторів від віртуальних артефактів через десеріалізацію JSON-конфігурацій. Крім того, реалізовано повноцінну економічну підсистему віртуального магазину із забезпеченням транзакційної цілісності даних.

Клієнтську частину програмної системи розроблено у форматі Single Page Application за допомогою фреймворку Angular. Під час створення фронтенду успішно застосовано новітні архітектурні підходи, зокрема ізольовані компоненти (Standalone Components) та механізм реактивного управління станом інтерфейсу (Signals). Візуальне оформлення застосунку було реалізовано за допомогою CSS-препроцесора SCSS та бібліотеки NES.css, що дозволило створити унікальний та цілісний користувацький досвід у стилістиці 8-бітних ретро-ігор, який органічно доповнює загальну концепцію гейміфікації.

Для забезпечення високого рівня надійності розробленого програмного забезпечення ключові бізнес-алгоритми було покрито модульними тестами з використанням фреймворку xUnit та інструментів імітації об'єктів Moq.

Загалом розроблений програмний комплекс є завершеним, повністю працездатним і відповідає всім заданим функціональним та нефункціональним вимогам. Результати роботи демонструють ефективність поєднання сучасних вебтехнологій із принципами гейм-дизайну для вирішення прикладних задач самоорганізації та продуктивності. Здобуті професійні навички проєктування, програмування та тестування, а також створений програмний код становлять повноцінну основу для подальшого розвитку проєкту та його успішного впровадження у практику персонального тайм-менеджменту.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Chou Y. Actionable Gamification: Beyond Points, Badges, and Leaderboards. Octalysis Media, 2015. 512 p.
2. Zichermann G., Cunningham C. Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps. O'Reilly Media, 2011. 208 p.
3. Еяль Н. На гачку. Як створити продукт, що привчає до споживання / пер. з англ. О. Асташової. Київ : Наш формат, 2017. 224 с.
4. Пасічник В. В., Жежнич П. І., Кравець Р. Б. Глобальні інформаційні системи та технології : підручник. Львів : Видавництво Львівської політехніки, 2014. 448 с.
5. Habitica: Gamify Your Life. Офіційний сайт Habitica. URL: <https://habitica.com/> (дата звернення: 08.05.2026).
6. Todoist: The to do list to organize work & life. Офіційний сайт Todoist. URL: <https://todoist.com/> (дата звернення: 08.05.2026).
7. Duolingo: The world's best way to learn a language. Офіційний сайт Duolingo. URL: <https://www.duolingo.com/> (дата звернення: 08.05.2026).
8. Gane C., Sarson T. Structured Systems Analysis: Tools and Techniques. New York : Improved System Technologies, 1979. 373 p.
9. Yourdon E. Modern Structured Analysis. Englewood Cliffs : Yourdon Press, 1989. 688 p.
10. UML Use Case Diagram Tutorial. Lucidchart. URL: <https://www.lucidchart.com/pages/uml-use-case-diagram> (дата звернення: 09.05.2026).
11. Freeman A. Pro ASP.NET Core 7. 10th Edition. New York : Apress, 2023. 1438 p.
12. C# для чайників. Мюллер Д. П., Сфер Ч., Семпф Б. 2022 р.
13. Entity Framework Core. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/ef/core/> (дата звернення: 12.05.2026).

14. JSON Web Token (JWT) Introduction. Офіційний сайт JWT. URL: <https://jwt.io/introduction> (дата звернення: 12.05.2026).
15. RFC 7519: JSON Web Token (JWT). IETF Datatracker. URL: <https://datatracker.ietf.org/doc/html/rfc7519> (дата звернення: 20.05.2026).
16. Common web application architectures. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures> (дата звернення: 13.05.2026).
17. Fain Y., Moiseev A. Angular Development with TypeScript. 2nd ed. Shelter Island : Manning Publications, 2018. 560 p.
18. Understanding Angular Signals: The New Era of Reactive Programming URL: <https://medium.com/@shivendra3549/understanding-angular-signals-the-new-era-of-reactive-programming-c4f641309668> (дата звернення: 12.05.2026).
19. Seshadri S. Angular: Up and Running: Learning Angular Step by Step. URL: https://books.google.com.ua/books?hl=uk&lr=&id=CAFeDwAAQBAJ&oi=fnd&pg=PT17&dq=+Seshadri+S.+Angular:+Up+and+Running:+Learning+Angular+Step+by+Step&ots=WaFJxUM_aj&sig=Ua4VktLjWGpjenDNz7x2ROc0VGU&redir_esc=y#v=onepage&q=Seshadri%20S.%20Angular%3A%20Up%20and%20Running%3A%20Learning%20Angular%20Step%20by%20Step&f=false (дата звернення: 12.05.2026).
20. NES.css: NES-style CSS Framework. GitHub. URL: <https://nostalgic-css.github.io/NES.css/> (дата звернення: 12.05.2026).
21. Берко А. Ю., Верес О. М., Пасічник В. В. Системи баз даних : підручник. Львів : Новий Світ-2000, 2018. 484 с. URL: <https://lib.ugi.edu.ua/bib/1134> (дата звернення: 12.05.2026).
22. xUnit.net: Unit testing tool for .NET. URL: <https://xunit.net/> (дата звернення: 13.05.2026).
23. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 352 p.

24. JSON serialization and deserialization in .NET. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-overview> (дата звернення: 14.05.2026).

25. Database design basics. Microsoft Support. URL: <https://support.microsoft.com/en-us/office/database-design-basics-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5> (дата звернення: 12.05.2026).

26. Object-Oriented Programming in C#. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/object-oriented-programming> (дата звернення: 10.05.2026).

27. JSON serialization and deserialization in .NET. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-overview> (дата звернення: 14.05.2026).

28. Introduction to Identity on ASP.NET Core. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity> (дата звернення: 14.05.2026).

29. Cross-Origin Resource Sharing (CORS). MDN Web Docs. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS> (дата звернення: 19.05.2026).

30. Enable Cross-Origin Requests (CORS) in ASP.NET Core. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/aspnet/core/security/cors> (дата звернення: 19.05.2026).

31. Choosing a testing strategy. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/ef/core/testing/choosing-a-testing-strategy> (дата звернення: 15.05.2026).

32. Moq: The most popular and friendly mocking framework for .NET. GitHub. URL: <https://github.com/devlooped/moq> (дата звернення: 15.05.2026).

33. Testing with EF Core In-Memory. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/ef/core/testing/in-memory> (дата звернення: 15.05.2026).

34. image-rendering - CSS: Cascading Style Sheets. MDN Web Docs. URL: <https://developer.mozilla.org/en-US/docs/Web/CSS/image-rendering> (дата звернення: 22.05.2026)

ДОДАТОК А

«Вебзастосунок планування задач з елементами ігрофікації та віртуальним аватаром»

НТУУ «КПІ ім. Ігоря Сікорського»

НН ІАТЕ

Кафедра ЦТЕ

Група ТР-25

Реалізація сервісу завдань

Мова програмування: С#

Аркушів 11

```

using AdvenTask.Core.DTO.QuestDTOs;
using AdvenTask.Core.Entities;
using AdvenTask.Core.Enums;
using AdvenTask.Core.ServiceContracts;
using AdvenTask.Infrastructure.DatabaseContext;
using Microsoft.EntityFrameworkCore;
using System.Text.Json;
namespace AdvenTask.Infrastructure.Services
{
    public class QuestService : IQuestService
    {
        private readonly ApplicationDbContext _context;
        public QuestService(ApplicationDbContext context)
        {
            _context = context;
        }
        #region QuestLines
        public async Task<List<QuestLineDTO>> GetQuestLinesAsync(Guid userId)
        {
            return await _context.QuestLines
                .AsNoTracking()
                .Where(ql => ql.UserId == userId)
                .Include(ql => ql.Quests)
                .Select(ql => new QuestLineDTO
                {
                    Id = ql.Id,
                    Title = ql.Title,
                    Quests = ql.Quests.Select(q => MapToQuestDTO(q)).ToList()
                })
        }
    }
}

```

```

        .ToListAsync();
    }

    public async Task<QuestLineDTO?> GetQuestLineByIdAsync(Guid id, Guid
userId)
    {
        var questLine = await _context.QuestLines
            .AsNoTracking()
            .Include(ql => ql.Quests)
            .FirstOrDefaultAsync(ql => ql.Id == id && ql.UserId == userId);
        if (questLine == null)
            return null;
        return new QuestLineDTO
        {
            Id = questLine.Id,
            Title = questLine.Title,
            Quests = questLine.Quests.Select(MapToQuestDTO).ToList()
        };
    }

    public async Task<QuestLineDTO> CreateQuestLineAsync(CreateQuestLineDTO
createDTO, Guid userId)
    {
        var questLine = new QuestLine
        {
            Id = Guid.NewGuid(),
            UserId = userId,
            Title = createDTO.Title
        };
        _context.QuestLines.Add(questLine);
        await _context.SaveChangesAsync();
    }

```

```

return new QuestLineDTO
{
    Id = questLine.Id,
    Title = questLine.Title,
    Quests = new List<QuestDTO>()
};
}

public async Task<(bool Succeeded, string Error)> UpdateQuestLineAsync(Guid id,
CreateQuestLineDTO updateDTO, Guid userId)
{
    var questLine = await _context.QuestLines
        .FirstOrDefaultAsync(ql => ql.Id == id && ql.UserId == userId);

    if (questLine == null)
        return (false, "QuestLine not found or access denied");

    questLine.Title = updateDTO.Title;
    await _context.SaveChangesAsync();

    return (true, string.Empty);
}

public async Task<(bool Succeeded, string Error)> DeleteQuestLineAsync(Guid id,
Guid userId)
{
    var questLine = await _context.QuestLines
        .FirstOrDefaultAsync(ql => ql.Id == id && ql.UserId == userId);

    if (questLine == null)
        return (false, "QuestLine not found or access denied");

```

```

        _context.QuestLines.Remove(questLine);
        await _context.SaveChangesAsync();
        return (true, string.Empty);
    }
}

#endregion

#region Quests

public async Task<(bool Succeeded, string Error, QuestDTO? Quest)>
CreateQuestAsync(CreateQuestDTO createDTO, Guid userId)
{
    var questLineExists = await _context.QuestLines
        .AnyAsync(ql => ql.Id == createDTO.QuestLineId && ql.UserId ==
userId);
    if (!questLineExists)
        return (false, "QuestLine not found or access denied", null);
    var quest = new Quest
    {
        Id = Guid.NewGuid(),
        QuestLineId = createDTO.QuestLineId,
        Title = createDTO.Title,
        Description = createDTO.Description,
        Deadline = createDTO.Deadline,
        Difficulty = createDTO.Difficulty,
        Tag = createDTO.Tag,
        IsCompleted = false
    };
    _context.Quests.Add(quest);
    await _context.SaveChangesAsync();
    return (true, string.Empty, MapToQuestDTO(quest));
}

```

```

    public async Task<(bool Succeeded, string Error, QuestDTO? Quest)>
UpdateQuestAsync(Guid id, UpdateQuestDTO updateDTO, Guid userId)
    {
        var quest = await _context.Quests
            .Include(q => q.QuestLine)
            .FirstOrDefaultAsync(q => q.Id == id && q.QuestLine.UserId == userId);
        if (quest == null)
            return (false, "Quest not found or access denied", null);
        quest.Title = updateDTO.Title;
        quest.Description = updateDTO.Description;
        quest.Deadline = updateDTO.Deadline;
        quest.Difficulty = updateDTO.Difficulty;
        quest.Tag = updateDTO.Tag;
        await _context.SaveChangesAsync();
        return (true, string.Empty, MapToQuestDTO(quest));
    }

    public async Task<(bool Succeeded, string Error)> DeleteQuestAsync(Guid id, Guid
userId)
    {
        var quest = await _context.Quests
            .Include(q => q.QuestLine)
            .FirstOrDefaultAsync(q => q.Id == id && q.QuestLine.UserId == userId);
        if (quest == null)
            return (false, "Quest not found or access denied");
        _context.Quests.Remove(quest);
        await _context.SaveChangesAsync();
        return (true, string.Empty);
    }

    public async Task<(bool Succeeded, string Error, QuestDTO? Quest)>
ToggleQuestStatusAsync(Guid id, bool isCompleted, Guid userId)

```

```

{
    var quest = await _context.Quests
        .Include(q => q.QuestLine)
        .FirstOrDefaultAsync(q => q.Id == id && q.QuestLine.UserId == userId);
    if (quest == null)
        return (false, "Quest not found or access denied", null);
    if (quest.IsCompleted == isCompleted)
        return (true, string.Empty, MapToQuestDTO(quest));
    quest.IsCompleted = isCompleted;
    quest.CompletedAt = isCompleted ? DateTime.UtcNow : null;
    var user = await _context.Users.FirstOrDefaultAsync(u => u.Id == userId);
    if (user != null)
    {
        var rewards = CalculateBaseRewards(quest.Difficulty);
        double finalXp = rewards.XP;
        double finalCoins = rewards.Coins;
        if (isCompleted)
        {
            bool isLate = quest.Deadline.HasValue && DateTime.UtcNow >
quest.Deadline.Value;
            if (!isLate)
            {
                var equippedArtifacts = await _context.UserArtifacts
                    .Include(ua => ua.Artifact)
                    .Where(ua => ua.UserId == userId && ua.IsEquipped)
                    .ToListAsync();
                (finalXp, finalCoins) = ApplyArtifactEffects(equippedArtifacts, quest,
finalXp, finalCoins);
            }
        }
    }
}

```

```

        user.Coins += (int)Math.Round(finalCoins);
        AddExperience(user, (int)Math.Round(finalXp));
    }
    else
    {
        user.Coins = Math.Max(0, user.Coins - rewards.Coins);
        user.Experience = Math.Max(0, user.Experience - rewards.XP);
    }
}
await _context.SaveChangesAsync();
return (true, string.Empty, MapToQuestDTO(request));
}
#endregion

#region Helpers
private static QuestDTO MapToQuestDTO(Quest q)
{
    return new QuestDTO
    {
        Id = q.Id,
        QuestLineId = q.QuestLineId,
        Title = q.Title,
        Description = q.Description,
        Deadline = q.Deadline,
        Difficulty = q.Difficulty,
        Tag = q.Tag,
        IsCompleted = q.IsCompleted,
        CompletedAt = q.CompletedAt
    };
}

```

```

private (int XP, int Coins) CalculateBaseRewards(Difficulty difficulty)
{
    return difficulty switch
    {
        Difficulty.Easy => (XP: 25, Coins: 10),
        Difficulty.Medium => (XP: 50, Coins: 25),
        Difficulty.Hard => (XP: 100, Coins: 60),
        _ => (XP: 25, Coins: 10)
    };
}

private (double XP, double Coins) ApplyArtifactEffects(List<UserArtifact>
equippedArtifacts, Quest quest, double currentXp, double currentCoins)
{
    double xpMultiplier = 1.0;
    double coinsMultiplier = 1.0;
    double xpFlat = 0;
    double coinsFlat = 0;
    foreach (var ua in equippedArtifacts)
    {
        if (string.IsNullOrEmpty(ua.Artifact.EffectsJson))
            continue;
        try
        {
            var effects =
JsonSerializer.Deserialize<List<ArtifactEffect>>(ua.Artifact.EffectsJson);
            if (effects == null)
                continue;
            foreach (var effect in effects)
            {

```

```

bool isApplicable = false;

switch (effect.TargetType)
{
    case "Global":
        isApplicable = true;
        break;
    case "Difficulty":
        if (effect.TargetValue != null && effect.TargetValue ==
quest.Difficulty.ToString())
            isApplicable = true;
        break;
    case "Tag":
        if (quest.Tag.HasValue && effect.TargetValue != null &&
effect.TargetValue == quest.Tag.Value.ToString())
            isApplicable = true;
        break;
}
if (!isApplicable)
    continue;
if (effect.RewardType == "XP")
{
    if (effect.Operator == "Multiplier")
        xpMultiplier *= effect.Value;
    else if (effect.Operator == "Flat")
        xpFlat += effect.Value;
}
else if (effect.RewardType == "Coins")
{

```

```

        if (effect.Operator == "Multiplier")
            coinsMultiplier *= effect.Value;
        else if (effect.Operator == "Flat")
            coinsFlat += effect.Value;
    }
}
}
catch
{
}
}

double finalXp = (currentXp * xpMultiplier) + xpFlat;
double finalCoins = (currentCoins * coinsMultiplier) + coinsFlat;
return (finalXp, finalCoins);
}

private void AddExperience(Core.Identity.ApplicationUser user, int xpToAdd)
{
    user.Experience += xpToAdd;
    while (true)
    {
        int xpRequired = GetXpRequiredForNextLevel(user.Level);
        if (user.Experience >= xpRequired)
        {
            user.Experience -= xpRequired;
            user.Level++;
        }
        else
        {
            break;
        }
    }
}

```

```

    }
}
}
private int GetXpRequiredForNextLevel(int currentLevel)
{
    return currentLevel * 100;
}
private class ArtifactEffect
{
    public string TargetType { get; set; } = string.Empty; // Global, Tag,
Difficulty
    public string? TargetValue { get; set; }
    public string RewardType { get; set; } = string.Empty; // XP, Coins
    public string Operator { get; set; } = string.Empty; // Multiplier, Flat
    public double Value { get; set; }
}
#endregion
}
}

```

ДОДАТОК Б

Апробація

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»**СУЧАСНІ ПРОБЛЕМИ
НАУКОВОГО ЗАБЕЗПЕЧЕННЯ
ЕНЕРГЕТИКИ:
БЕЗПЕКА, СТІЙКІСТЬ,
ІТ ТА ЕКОЛОГІЧНИЙ МОНІТОРИНГ**XXIII Міжнародна науково-практична конференція
молодих вчених та студентів

До 40-річчя Чорнобильської катастрофи

17–22 квітня 2026 року, м. Київ



Київ-2026

УДК 004.4

¹ Бакалаврант 4 курсу Білоус О. В.

² Доц., к.е.н. Сегеда І. В.

<https://scholar.google.com.ua/citations?hl=en&user=cTob-AwAAAAJ>

³ КПІ ім. Ігоря Сікорського

РОЗРОБКА ВЕБ-ЗАСТОСУНКУ ДЛЯ ПЛАНУВАННЯ ЗАДАЧ З ЕЛЕМЕНТАМИ ІГРОФІКАЦІЇ ТА ВІРТУАЛЬНИМ АВАТАРОМ

Постановка проблеми та її актуальність. У сучасному світі з вільним доступом до цифрового середовища, користувачі все частіше стикаються з проблемою дефіциту уваги та втрати мотивації при використанні традиційних методів планування завдань (To-Do lists). Звичайні цифрові або ж паперові планери швидко стають рутинною. Впровадження елементів ігрофікації та механік утримання інтересу через віртуального аватара є одним з варіантів вирішення питання залученості користувачів.

Аналіз останніх досліджень. Остання дослідження підтверджують, що впровадження ігрових елементів у неігрові процеси (такі як навчання або планування) дозволяє ефективно мотивувати користувача через систему віртуальних нагород та індикаторів прогресу [1].

Проекти, до прикладу Habitica, демонструють успішне використання RPG-елементів у тайм-менеджменті, що доводить перспективу використання даної стратегії для розробки цифрових продуктів.

Метою даної роботи є розробка архітектури та реалізація веб-застосунку, який інтегрує механіки виконання запланованих завдань з ігор тематики RPG у процес планування задач, заохочуючи користувача виконувати поставлені задачі отриманням віртуальної нагороди.

Основна частина. Архітектура веб-застосунку побудована на принципах клієнт-серверної взаємодії з використанням принципу SPA (Single Page Application).

Архітектурними рівнями є:

Клієнтська частина реалізована з використанням фреймворку Angular, що базується на компонентно-орієнтованій архітектурі. Кожен елемент інтерфейсу (список задач, панель аватара, ігровий магазин) є окремим ізольованим модулем.

Для управління станом додатку та взаємодії з API в проєкті передбачені сервіси: «Сервіс автентифікації/авторизації»; «Сервіс управління завданнями»; «Сервіс артефактів» та «Сервіс аватарів».

Серверна частина – виступає в ролі REST API, обробляє запити від клієнта та реалізує основну бізнес-логіку додатку. В ній реалізовані контролери, що приймають HTTPS-запити, валідують отримані дані та повертають відповіді у форматі JSON. В склад серверної частини входить Шар логіки. Він відповідає за операції обчислення досвіду та ігрової віртуальної валюти, з урахуванням отриманих штрафів та використанням допоміжних модифікаторів заробітку валюти.

Для забезпечення зручної та безпечної взаємодії з базою даних, передбачається використання Entity Framework, як ORM-прошарок між різними рівнями архітектури.

Рівень даних – реалізований через реляційну СУБД SQL Server. Концептуальну схему базу даних представлено на рис. 1.

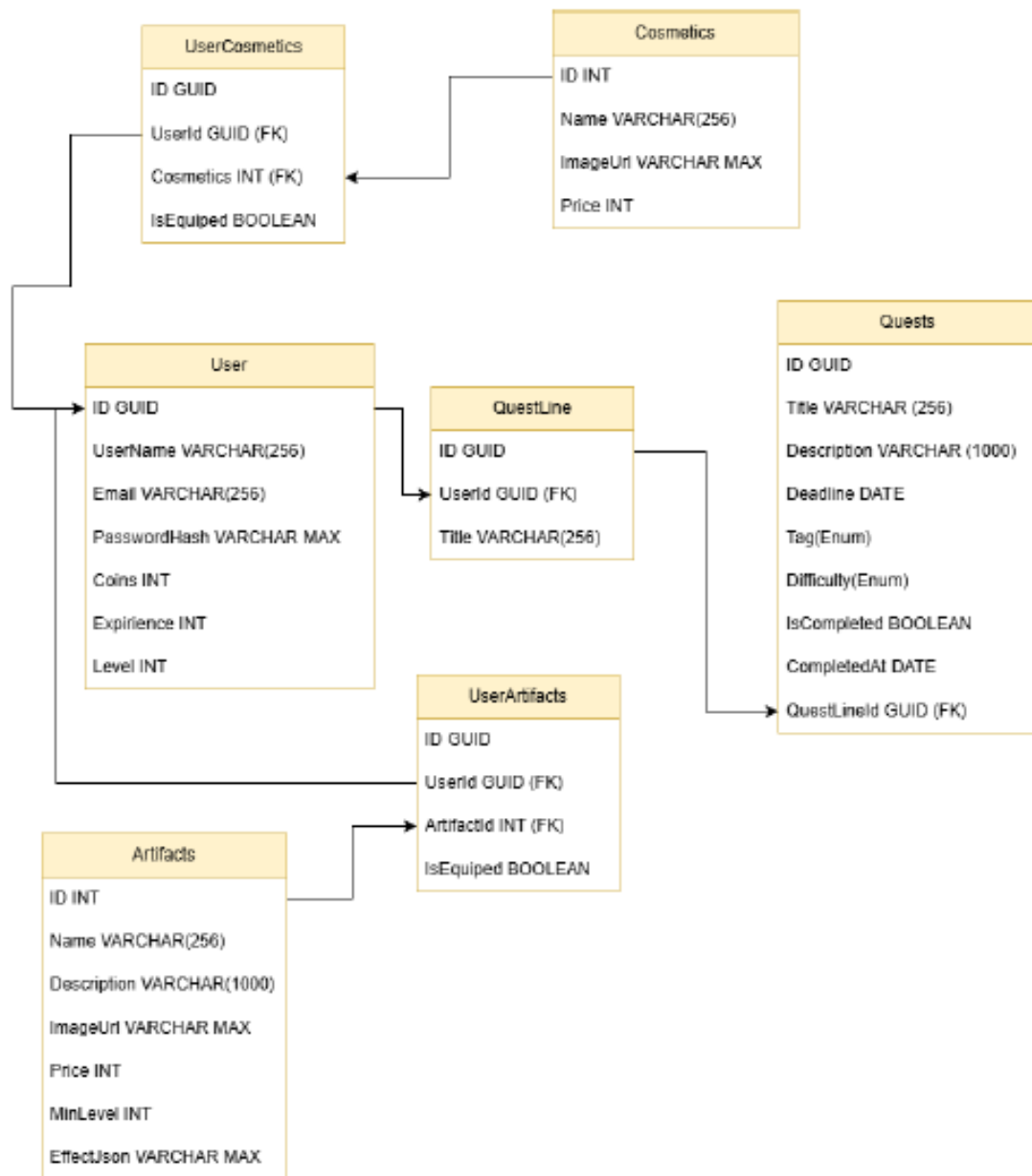


Рисунок 1 – Концептуальна схема бази даних веб-додатку

Протокол взаємодії. Зв'язок між Angular-клієнтом та ASP.NET-сервером відбувається через асинхронні HTTPS-запити.

Кожна створена задача при виконанні приносить певну кількість віртуальної ігрової валюти (Coins) та одиниць досвіду (XP). Кількість заробленої валюти залежить від рівня складності(Легка, Середня, Складна) поставленої задачі та досягнення певного рівня досвіду. За зароблену валюту можна придбати віртуальні аватари або модифікатори, які збільшують об'єм отриманих винагород.

Функціональна модель системи надає користувачу повний контроль над життєвим циклом завдань. Користувач має доступ до персонального списку задач з можливістю їх створення, редагування, виконання та видалення. Реалізовано право користувача за зароблену валюту придбати покращення для власного облікового запису: новий аватар чи артефакт для збільшення ефективності заробітку віртуальних ресурсів. Всі права доступу валідуються серверною частиною через протокол HTTPS.

Користувач має виключне право на доступ до власного профілю після проходження процедури аутентифікації. Всі операції захищені на рівні бекенду, що унеможливує доступ сторонніх осіб до персональних списків задач або маніпуляції з ігровим балансом інших учасників.

Через інтерфейс користувач має можливість переглядати наступну статистику на сторінці користувача: рівень досвіду, кількість заробленого досвіду, куплені та екіпировані модифікатори прогресу, аватар, який використовується в поточний період.

Для реалізації клієнтської частини обрано фреймворк Angular у поєднанні з мовою TypeScript. Використання Angular дозволяє створювати масштабовані веб-застосунки з чіткою структурою коду, що значно спрощує підтримку складних ігрових механік та анімацій [2]. Для обробки бізнес-логіки застосунку обрано ASP.NET Core, що дозволяє реалізувати надійну архітектуру REST API для взаємодії з клієнтською частиною на Angular. Використання даної платформи сприяє створенню безпечних та масштабованих інформаційних систем [3].

У розробленому веб-застосунку серверна частина відповідає за валідацію виконаних завдань, обрахунок ігрової валюти та збереження прогресу користувача, що гарантує цілісність даних та захист від некоректних маніпуляцій з ігровими показниками.

Для збереження структурованих даних про користувачів, їхні списки задач та поточні характеристики віртуального супутника застосовано реляційну систему керування базами даних SQL Server.

Алгоритм взаємодії системи працює наступним чином:

1. Користувач створює задачу, параметри якої (назва, опис, складність, термін, сфера життя) зберігаються в SQL Server та обраховується винагорода за виконання.
2. При відмітці про виконання, Angular-клієнт надсилає запит на сервер, де логіка на C# нараховує зароблені ресурси.
3. Своєчасне виконання задач призводить до накопичення валюти та досвіду, тоді як порушення кінцевих термінів виконання зменшує винагороду через систему штрафу.

Реалізація програмного продукту поділена на окремі етапи реалізації:

1. Визначення вимог, функціоналу, проєктування реляційної бази даних у 3NF та структури проєкта;
2. Розробка серверної логіки на ASP.NET Core для обрахунку ігрових показників, створення моделей даних, API-контролерів та системи автентифікації користувачів;
3. Побудова інтерактивного SPA-інтерфейсу на Angular;
4. Тестування та налагодження роботи додатку.

Висновки. Розробка додатку-планувальника з елементами ігрофікації та віртуальним аватаром дозволяє змінити підхід сприйняття процесу планування з обов'язку в інтерактивно-ігровий досвід. Подальша розробка програмного продукту може бути спрямована на впровадження соціальних механік взаємодії між користувачами та адаптивних алгоритмів для автоматичного коригування складності завдань залежно від успішності гравця.

Перелік посилань:

1. Матвійчук О. О. Гейміфікація як невід'ємний чинник підвищення ефективності елементів дистанційного навчання. URL: <https://eprints.zu.edu.ua/17143/1/357.pdf> (дата звернення: 26.02.2026).
2. Angular. Робота з сервісами. Блог Hillel. URL: <https://blog.ithillel.ua/videos/angular-kak-rabotat-s-servisami> (дата звернення: 26.02.2026).
3. Аналіз сучасного фреймворка ASP.NET CORE для WEB-додатків. URL: <https://openarchive.nure.ua/entities/publication/f046f6e2-3e03-41cd-a6d1-5974781f8abf> (дата звернення: 26.02.2026).