МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ПРОГРАМНОГО КОМПЛЕКСУ ТА МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ ДЛЯ ОПТИМІЗАЦІЇ ПРОЦЕСУ ПЕРЕКЛАДУ МОДИФІКАЦІЙ PARADOX»**

Виконала: студентка 2 курсу, групи 8.1214-з
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
К.О. Шкатула
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Мухін В.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Шкатулі Карині Олександрівні
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка програмного комплексу мікросервісної архітектури для оптимізації процесу перекладу модифікацій Paradox

керівник роботи Мухін Віталій Вікторович, к.т.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 28 » лютого 2025 року № 327-с

2. Строк подання студентом роботи 28.11.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та постановка задачі.
2. Проєктування архітектури системи.
3. Реалізація інформаційної системи.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 14.03.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	28.03.2025	
2.	Збір вихідних даних.	18.04.2025	
3.	Обробка методичних та теоретичних	20.06.2025	
	джерел.		
4.	Розробка першого та другого розділу.	05.09.2025	
5.	Розробка третього розділу.	17.11.2025	
6.	Оформлення та нормоконтроль	24.11.2025	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	12.12.2025	

Студент _____ К.О. Шкатула
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.В. Мухін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка програмного комплексу та мікросервісної архітектури для оптимізації процесу перекладу модифікацій Paradox»: 86 с., 45 рис., 6 табл., 27 джерел, 3 додатка.

БАЗА ДАНИХ, ВЕБРОЗРОБКА, КОМП'ЮТЕРНІ ІГРИ, ПЕРЕКЛАДИ, СЕРВІС ПЕРЕКЛАДУ, NODEJS.

Об'єкт дослідження – сервіс перекладу модів Paradox.

Мета роботи: розробити сервіс перекладу с використання мікросервісної архітектури.

Методи дослідження – дослідження та імплементація мікросервісної архітектури, методи програмної інженерії, методи аналізу даних, методи проєктування, конструювання програмного забезпечення.

У кваліфікаційній роботі розглянуто процес розробки WEB сервісу. Проведено аналіз предметної області, технологій та інструментів для вирішення поставленого завдання, розроблені наступні діаграми: прецедентів, діяльності, компоненти, класів та розгортання, компоненти, класів та розгортання, спроєктована структура додатка, а також база даних. За результатом виконання кваліфікаційної роботи, реалізований вебсервіс для перекладу модифікацій ігор Paradox Interactive на основі мікросервісної архітектури з використанням NodeJS та NextJS в якості фронтенд фреймворка. Навички, отримані під час розробки WEB сервісу, можуть бути застосовані на реальних проєктах.

SUMMARY

Master’s qualifying paper “Development of a Software Complex and Microservice Architecture to Optimize the Translation Process of Paradox Modifications”: 86 pages, 45 figures, 6 tables, 27 references, 3 supplements.

DATABASE, WEB DEVELOPMENT, COMPUTER GAMES, TRANSLATIONS, TRANSLATION SERVICE, NODEJS.

The object of the study is a translation service for Paradox modifications.

The aim of the study is to develop a translation service using a microservice architecture.

The methods of research are the investigation and implementation of microservice architecture, software engineering methods, data analysis methods, and methods for designing and constructing software.

In this qualification work, the process of developing a WEB service is examined. An analysis of the subject area, technologies, and tools for solving the assigned task has been carried out. The following diagrams were developed: use case, activity, component, class, and deployment diagrams. The application structure as well as the database were designed. As a result of the qualification work, a web service was implemented for translating Paradox Interactive game modifications, based on a microservice architecture using NodeJS and NextJS as the frontend framework. The skills acquired during the development of the WEB service can be applied to real-world projects.

ЗМІСТ

Завдання на кваліфікаційну роботу……2
Реферат 4
Summary 5
Скорочення та умовні познаки 8
Вступ..... 9
1 Аналіз предметної області та постановка завдання..... 11
1.1 Огляд індустрії комп’ютерних ігор..... 11
1.2 Аналіз існуючих рішень до автоматизованого перегляду модифікацій 18
1.3 Обґрунтований вибір технологій..... 23
1.3.1 JavaScript (TypeScript)..... 24
1.3.2 Середовище Node.js 26
1.3.3 Redis (Remote Dictionary Server)..... 26
1.3.4 Мікросервісна архітектура..... 28
1.3.5 DeepL API 30
1.3.6 OpenAI API 32
1.4 Висновки до першого розділу..... 34
2 Проєктування архітектури системи..... 36
2.1 Архітектура додатку: взаємодія компонентів та потоки даних 36
2.1.1 JavaScript та Node.js: переваги для серверної розробки..... 37
2.1.2 Redis для кешування та черг повідомлень 39
2.1.3 Масштабованість мікросервісної архітектури..... 42
2.2 Структура файлів локалізації Paradox-ігор та їх обробка..... 42
2.3 Структура бази даних 43
2.4 Діаграма прецедентів (Use Case Diagram)..... 44
2.5 Діаграма діяльності (Activity Diagram)..... 46
2.6 Діаграма компонентів (Component Diagram) 49
2.7 Діаграма класів (Class Diagram) 51

2.8 Діаграма розгортання (Deployment Diagram) 56
2.9 Висновки до другого розділу 57
3 Реалізація інформаційної системи 59
3.1 Опис розробленого програмного продукту 59
3.2 Реалізація основних модулів та функціоналу 61
3.3 Налаштування Docker контейнерів для розгортання 69
3.4 Тестування функціоналу системи 71
3.5 Висновки до третього розділу 73
Висновки 75
Перелік посилань 77
Додаток А Основний докер файл 79
Додаток Б Лістинг коду файлу для перекладу 84
Додаток В Схема бази даних 86

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД	База даних
ПК	Персональний комп’ютер
СУБД	Система управління базами даних
API	Application Programing Interface
DLC	Downloadable content
GPT	Generative pre-trained transformer
HTTP	Hypertext Transfer Protocol
ORM	Object-Relational Mapping
YAML	Yet Another Markup Language

ВСТУП

В сучасному світі ігрова індустрія продовжує стрімко розвиватися, залучаючи мільйони гравців з усього світу. Особливий інтерес представляють глобальні стратегічні ігри від компанії Paradox Interactive, такі як Europa Universalis, Crusader Kings, Hearts of Iron та Stellaris, які відомі своїми масштабними ігровими можливостями та підтримкою гравців у створенні модифікацій (модів). Модифікації дозволяють користувачам змінювати різні аспекти ігрового процесу, додаючи нові сценарії, культури, механіки та інші елементи. Однак одним із ключових викликів при створенні модифікацій є локалізація текстових даних для гравців різних мовних груп.

Апробація результатів дослідження: XVI Всеукраїнська, XXIII регіональна наукова конференція молодих дослідників «Актуальні проблеми математики та інформатики» 24 – 25 квітня 2025 рокі на базі ЗНУ, м. Запоріжжя, Україна. (публікація тез на тему «Застосування технологій машинного навчання для автоматизації та покращення якості перекладу ігрового контенту»).

Метою цієї кваліфікаційної роботи магістра є створення інструменту, що забезпечить користувачам зручний та ефективний процес перекладу текстових файлів модифікацій, зберігаючи при цьому точність та контекст оригіналу. В рамках роботи буде розроблено додаток, що автоматично адаптуватиме переклади з урахуванням специфічних термінів і механік ігор, забезпечуючи швидку та якісну локалізацію.

План кваліфікаційного проєкту:

- необхідність автоматизації локалізації модів для ігор Paradox Interactive:
- аналіз актуальності рішень для локалізації модифікацій до ігор Paradox;
- дослідження ком’юніті розробників та гравців модів Paradox Interactive, їхніх потреб у швидкій локалізації;
- огляд існуючих інструментів для перекладу модів, обмежень і ризиків встановлюваних додатків, що працюють лише на Windows;

– новизна проєкту: створення першого вебдодатку для локалізації модів з використанням сучасних API (DeepL та GPT).

Вибір фреймворку, бази даних та інструментарію для розробки:

– огляд і аналіз різних фреймворків для розробки вебдодатків, обґрунтування вибору Next.js;
– аналіз варіантів для керування чергами запитів і вибір Redis для оптимальної обробки перекладів в реальному часі;
– вибір API для перекладу (DeepL API) та покращення якості текстів (OpenAI API).

Огляд систем контролю версій, хмарних платформ і додаткових інструментів для підтримки розробки.

Архітектура та моделювання системи:

– проєктування архітектури вебдодатку на основі мікросервісної моделі;
– визначення функціональних компонентів системи: обробка файлів модів, робота з чергами Redis, інтеграція з API перекладу.

Імплементація та тестування вебдодатку:

– реалізація основного функціоналу вебдодатку: завантаження файлів локалізацій модів, управління чергами через Redis, інтеграція з DeepL та OpenAI API;
– тестування різних сценаріїв використання додатку: локалізація модів різних ігор, робота з великими обсягами даних, обробка змінних та спеціальних символів;
– інтеграція і тестування всіх компонентів у єдину систему.

Написання звіту та технічної документації.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Огляд індустрії комп’ютерних ігор

Ігри компанії Paradox Interactive є унікальним явищем у світі комп’ютерних стратегій. На відміну від більшості інших проєктів, орієнтованих на одного користувача, ці ігри не лише не втрачають своєї популярності з плином часу, але й, навпаки, зберігають високий рівень залученості гравців навіть через кілька років після релізу. Така ситуація є винятковою для індустрії, де більшість одно користувацьких ігор швидко втрачають аудиторію після проходження основної кампанії. В основі цього феномена лежить надзвичайно потужна система підтримки модифікацій, що дозволяє гравцям створювати та додавати новий контент, значно розширюючи можливості гри.

Модифікації (моди) грають ключову роль у підтримці інтересу до ігор Paradox Interactive. Вони дають можливість користувачам отримати новий ігровий досвід, додаючи нові сценарії, культури, нації, механіки або навіть повністю змінюючи правила гри. Саме завдяки модам ігри на кшталт Europa Universalis IV, Hearts of Iron IV, Crusader Kings III та Stellaris продовжують залишатися актуальними та популярними серед гравців багатьох років поспіль. Гравці можуть завантажити новий мод, який змінить ігровий процес таким чином, що гра сприймається зовсім по-іншому, навіть після багатьох годин попередньої гри. Важливість модифікацій підкреслюється й тим, що багато з них мають власну багатотисячну аудиторію шанувальників та активно підтримуються розробниками.

Однією з головних переваг модифікацій є їхня здатність адаптувати гру під різні культурні та мовознавчі середовища. Зокрема, велике значення має локалізація модів для гравців з неангломовних регіонів, таких як Китай, Туреччина, країни СНД тощо. Спільноти з цих регіонів мають значний вплив на загальну популярність ігор Paradox, тому важливість доступу до контенту

рідною мовою важко переоцінити. Відсутність якісної локалізації може стати значною перешкодою для гравців, оскільки ігри Paradox відомі своїм великим обсягом тексту, специфічними термінами та складними ігровими механіками, які без локалізації стають менш доступними.

Зважаючи на це, створення інструментів для локалізації модів стає вкрай важливим аспектом підтримки спільноти та подальшого розвитку ігор Paradox Interactive.

Статистика HoI 4 та порівняння. Ігри компанії Paradox Interactive є унікальним явищем у світі стратегічних ігор, маючи одну з найвідданіших аудиторій у індустрії. Більшість однокористувацьких проєктів з часом втрачають свою популярність, коли гравці завершують сюжетні лінії або вичерпують всі ігрові можливості. Однак ігри Paradox демонструють зовсім інший тренд: їхня спільнота не лише залишається активною, але й постійно зростає, навіть через кілька років після релізу. Це відбувається завдяки активній підтримці модифікацій, які надають нові можливості та дозволяють гравцям щоразу відкривати гру з іншого боку.

Моддінг є наріжним каменем успіху таких ігор, як Europa Universalis IV, Crusader Kings III, Hearts of Iron IV та Stellaris. Ці ігри створені з відкритою структурою, що дозволяє користувачам вносити зміни в ігрові механіки, додавати нові фракції, сценарії, механіки та навіть переписувати історичні події. Наприклад, у Hearts of Iron IV, грі про глобальні конфлікти, користувачі можуть створювати модифікації, які повністю змінюють політичний та військовий сценарій, відтворюючи альтернативні реальності. Завдяки цьому гравці можуть грати знову й знову, кожного разу отримуючи унікальний ігровий досвід.

Для розуміння важливості модифікацій, розглянемо приклад Hearts of Iron IV, яка була випущена 6 липня 2016 року. Незважаючи на те, що з моменту релізу минуло більше ніж шість років, гра продовжує отримувати нові доповнення (DLC) та патчі від розробників, а спільнота моддерів забезпечує гравців безперервним потоком нового контенту. Статистика Steam свідчить, що навіть через роки після виходу, середня кількість одночасних гравців залишається

стабільно високою, що є винятковим досягненням для однокористувацьких стратегій (див. рис. 1.1).



Рисунок 1.1 – Статистика онлайну Hoi4

Видно зі статистики, гра не втрачала популярності з моменту релізу, а лише набирала її. На відміну подібних їй по жанру, сеттінгу та ціновій політиці. Нижче приведено порівняння з іншими іграми: Age of Wonders 4, Endless Space 2 та Men of War 2 (див. рис. 1.2).

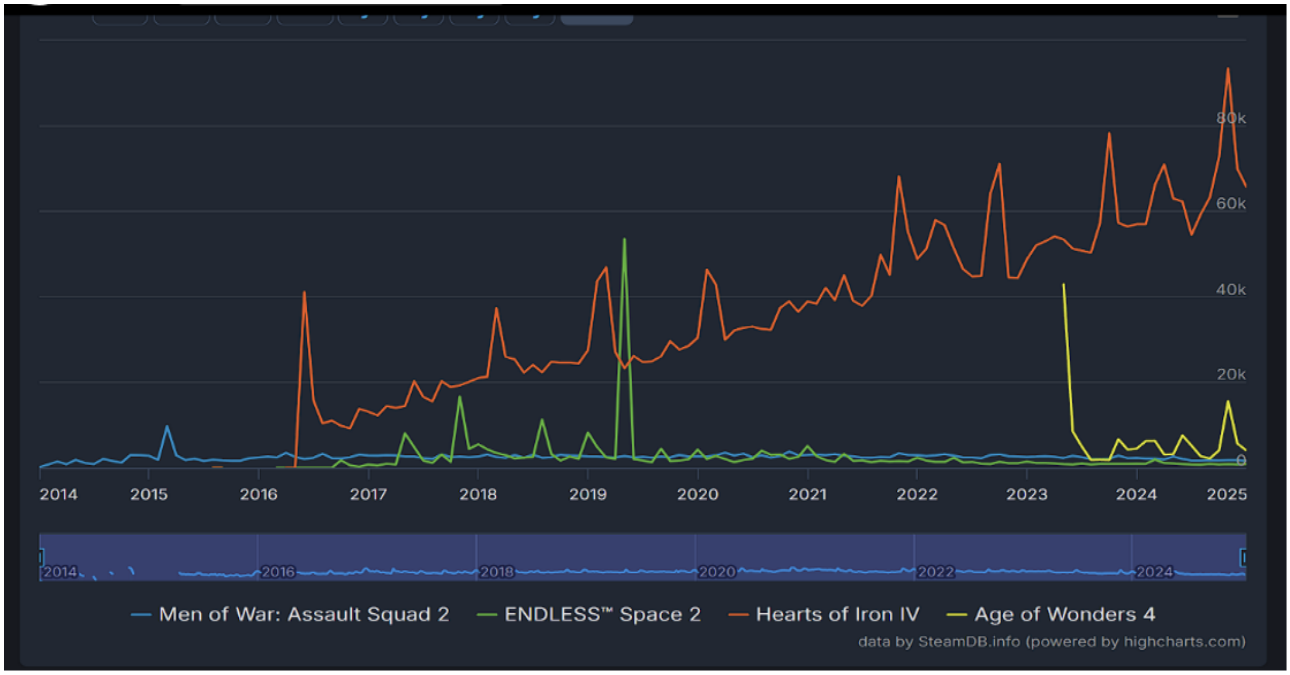


Рисунок 1.2 – Порівняння активного онлайну

Як видно з графіку, ці ігри мають кардинально різний онлайн. Щоб довести, що саме підтримка модінгу має кардинальне значення, можна порівняти інші параметри.

Підтримка контентом від офіційних розробників є ключовим елементом для утримання та зростання активної аудиторії гри. Регулярні DLC та контент-патчі відіграють важливу роль у продовженні життєвого циклу проєкту, адже вони додають нові механіки, режими та покращують загальний ігровий досвід. Команда розробників Paradox Interactive регулярно випускає доповнення для своїх ігор, що дозволяє не лише підтримувати інтерес з боку гравців, але й залучати нових користувачів. Це створює можливість порівняти їхній підхід із іншими іграми від розробників, які також активно підтримують свої проєкти після релізу. У таблиці нижче порівняно онлайн-активність кількох популярних ігор: Hearts of Iron IV, Age of Wonders 4, Endless Legend та Pathfinder: Wrath of the Righteous (див. рис. 1.3). Усі ці проєкти активно підтримувалися додатковим контентом, однак не всім з них вдалося зберегти зростання аудиторії на довготривалій основі.

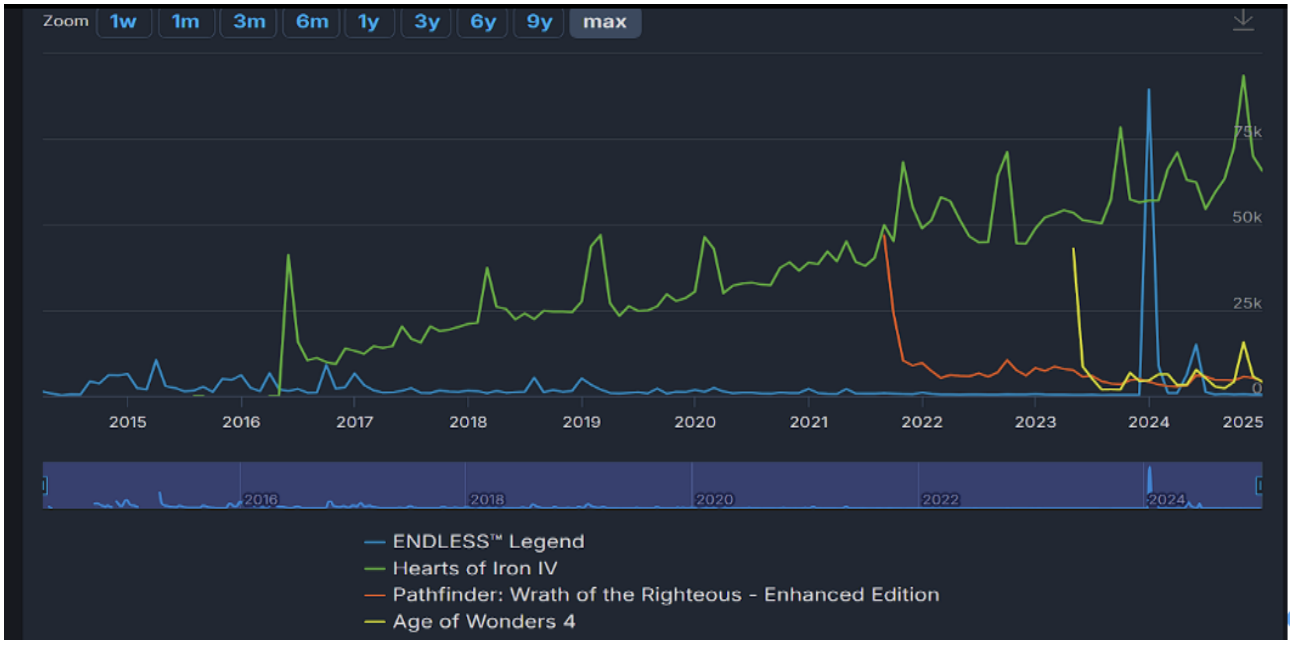


Рисунок 1.3 – Порівняння активного онлайну ігри з великими контент-патчами

Як видно з графіка, постійна підтримка контентом не завжди гарантує стабільне зростання чи навіть утримання аудиторії. У той час як такі ігри, як

Endless Legend та Pathfinder: Wrath of the Righteous, зіткнулися зі значним зниженням активності гравців навіть при регулярних оновленнях, Hearts of Iron IV продовжувала нарощувати аудиторію протягом усієї своєї історії. Це демонструє, що хоча кількість і регулярність доповнень важливі, інші фактори, такі як глибина геймплею, спільнота гравців та баланс нововведень, мають вирішальне значення для тривалого успіху гри.

Ігри від Paradox Interactive, як правило, належать до жанру складних 4X стратегій, які відрізняються великою кількістю параметрів, механік і глибоким плануванням. Ці проєкти охоплюють різні сфери управління: економіку, дипломатію, військову стратегію, дослідження нових територій та багато іншого. Водночас, їх часто називають «симуляторами контурних карт», оскільки значну частину часу гравці проводять, аналізуючи карту міста, світу або навіть галактики. Такий підхід робить ігри Paradox унікальними за рівнем деталізації та контролю, що особливо цінується гравцями, які шукають глибокий стратегічний досвід [2].

З огляду на складність жанру, коректним буде порівняти Hearts of Iron IV з іншими відомими 4X стратегіями, які також ставлять гравців перед викликами багатошарового управління та планування. Для цього вибрано такі ігри, як Sid Meier’s Civilization VI, Humankind та Age of Wonders 3, які мають спільні риси та подібну аудиторію (див. рис. 1.4).

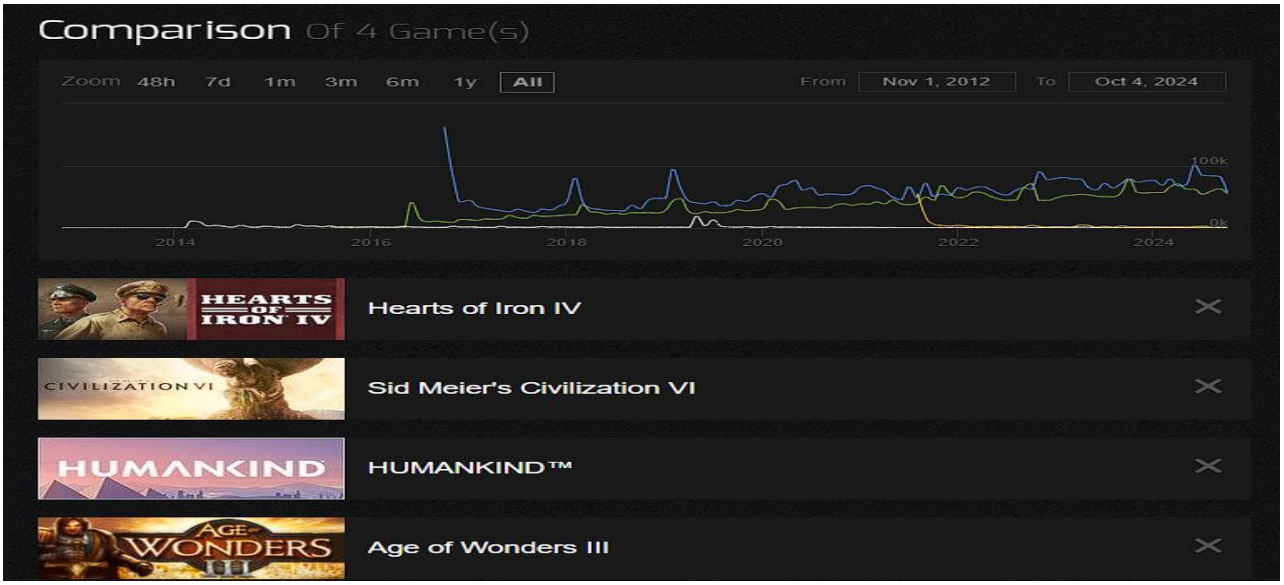


Рисунок 1.4 – Порівняння активного онлайну ігор одного жанру

Як видно з наведених даних, усі ігри цього жанру переживали зниження активного онлайну після початкового релізу. Особливо показовим є приклад Sid Meier’s Civilization VI, яка на момент виходу мала вражаючий онлайн у 160 тисяч гравців одночасно. Однак, вже через рік після релізу, онлайн гри знизився, і вона почала демонструвати показники, близькі до Hearts of Iron IV, яка стабільно трималася на рівні приблизно 20 тисяч гравців.

Тема «Другої світової війни» – один із найпопулярніших і найчастіше використовуваних у відеоіграх, оскільки вона надає гравцям можливість пережити та керувати однією з найбільш знакових і складних подій в історії людства. Цей сеттінг захоплює гравців, пропонуючи глибокі стратегії, реконструкцію історичних подій і можливість альтернативного розвитку подій.

У зв’язку з цим, доречно порівняти Hearts of Iron IV з іншими іграми, які також базуються на тематиці «Другої світової війни», щоб оцінити, як аудиторія реагує на різні підходи до цього історичного періоду.

У таблиці нижче наведено порівняння онлайн-активності таких ігор, як Company of Heroes 2, Men of War та Call to Arms: Ostfront (див. рис. 1.5). Ці ігри, хоч і відрізняються за геймплейними механіками та стилем подачі подій, мають спільну тематику і конкурують за увагу тієї самої аудиторії, що й Hearts of Iron IV.

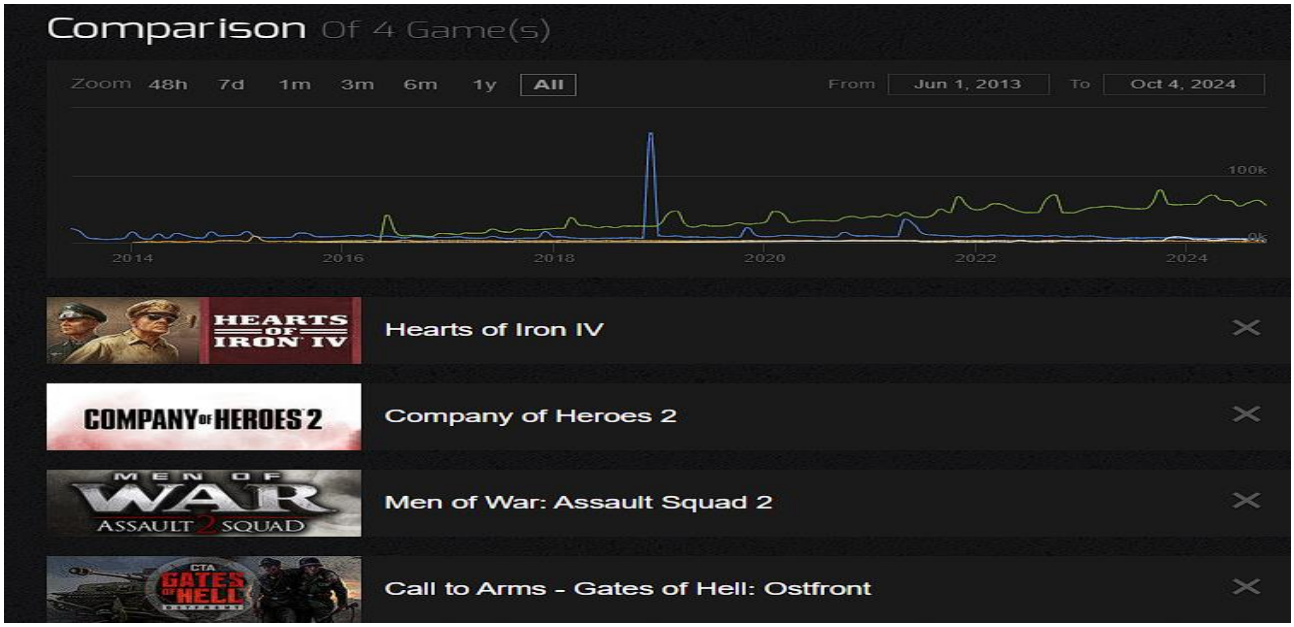


Рисунок 1.5 – Порівняння активного онлайну ігор одного жанру

Як показано на графіках, ситуація з онлайн-активністю ігор на тему: «Другої світової війни» демонструє схожі тенденції. Після виходу більшість із них поступово втрачали популярність, а значне зростання аудиторії відбувалося лише під час акцій та проведення «безкоштовних вихідних». Це особливо помітно у випадках Company of Heroes 2 та Men of War, які досягали своїх піків здебільшого за рахунок знижок і тимчасової безкоштовної доступності.

Отже, можна стверджувати, що саме активна підтримка спільноти моддерів є однією з ключових причин стабільної популярності Hearts of Iron IV. Створення великої кількості фанатського контенту забезпечує постійний інтерес до гри, навіть через кілька років після її випуску. Це дозволяє гравцям відчути нові враження, відкривати додаткові сценарії, нові механіки або покращення геймплею, що подовжує життєвий цикл гри.

Подібну ситуацію можна спостерігати і в інших популярних іграх, які мають сильну підтримку від моддерів. У таблиці нижче наведено порівняння онлайн-активності таких ігор, як Skyrim, Fallout: New Vegas та Fallout 4 (див. рис. 1.6), де спільнота моддерів також відіграє значну роль у підтримці інтересу до проєктів.

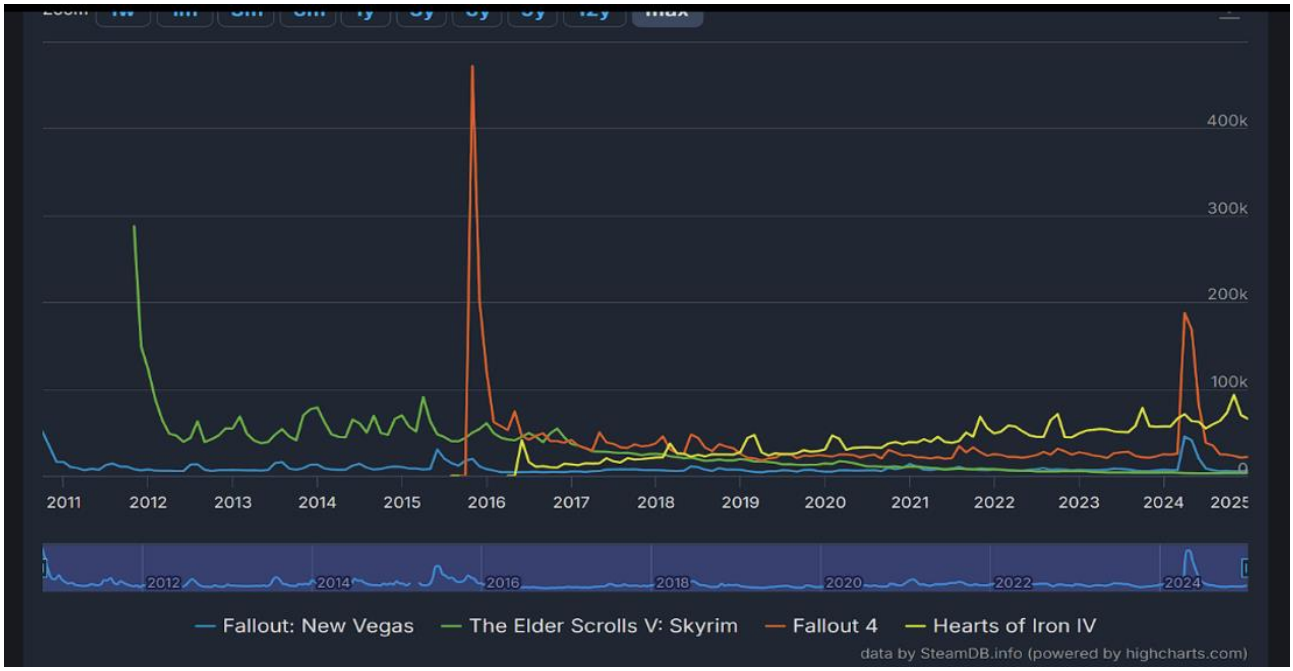


Рисунок 1.6 – Порівняння активних онлайн-ігор з активною модераторською спільнотою

Як видно з графіків, усі ці ігри мають схожу тенденцію: активна спільнота продовжує створювати величезну кількість модифікацій, що дозволяє цим іграм, орієнтованим на одного гравця, утримувати стабільну аудиторію протягом тривалого часу. На відміну від багатьох інших ігор, ці проєкти зберігають свою актуальність завдяки тому, що гравці можуть безперервно відкривати для себе новий контент. Це підкреслює важливість спільноти у збереженні довгострокового успіху гри. Постійний потік модифікацій, які доповнюють або повністю змінюють геймплей, створює унікальний досвід, що не дозволяє грі «застаріти» і стимулює гравців повертатися до неї знову і знову.

1.2 Аналіз існуючих рішень до автоматизованого перегляду модифікацій

Після аналізу впливу спільноти на підтримку популярності гри, важливо дослідити саму спільноту, її розміри, активність та продуктивність. Спільнота гравців Hearts of Iron IV та інших ігор від Paradox Interactive є однією з найбільш активних, згуртованих та продуктивних серед ігрових спільнот (див. рис. 1.7). Величезна кількість контенту, який створюється користувачами, вражає своєю різноманітністю та масштабами, що підтверджується даними з майстерні Steam.



Рисунок 1.7 – Головна сторінка майстерні

Як видно з майстерні Steam, кількість модифікацій для Hearts of Iron IV перевищує 130 тисяч. Серед них є як масштабні модифікації, які повністю змінюють гру, додаючи нові нації, сценарії та механіки, так і невеликі моди, які змінюють лише окремі елементи гри, наприклад, портрети персонажів чи інтерфейс. Окремо слід відзначити групу модифікацій, присвячених перекладу інших модів – таких налічується понад 4 тисячі, що свідчить про глобальну зацікавленість у доступності контенту різними мовними спільнотами.

Аномальна продуктивність спільноти Hearts of Iron IV викликає питання: чому саме ця гра залучає настільки велику кількість активних моддерів? Для порівняння, Sid Meier’s Civilization VI, маючи в середньому в півтора рази більше гравців, демонструє лише близько 30 тисяч модифікацій у своїй майстерні Steam. Це значно менше, ніж у Hearts of Iron IV, що вказує на унікальні особливості спільноти Paradox Interactive (див. рис. 1.8).

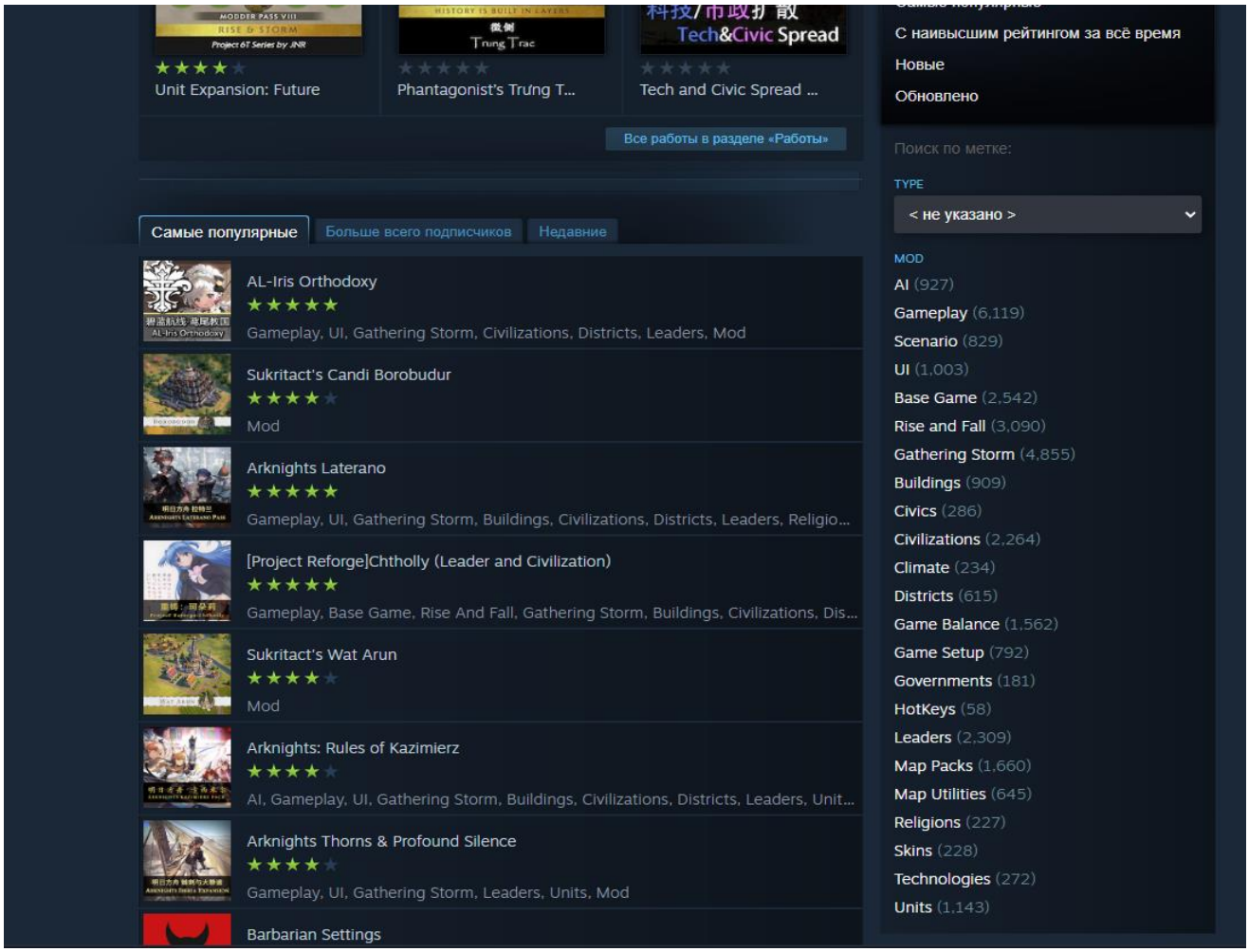


Рисунок 1.8 – Головна сторінка майстерні Civilization

Причин для такої виняткової продуктивності спільноти моддерів декілька, і всі вони тісно пов’язані з особливостями підходу Paradox Interactive до своїх ігор.

По-перше, ігри Paradox Interactive пропонують ідеальні умови для модифікації завдяки потужним інструментам та гнучким можливостям, які дозволяють змінювати практично всі аспекти гри. Компанія надає гравцям доступ до розширених інструментів розробки, що дозволяє створювати моди будь-якої складності – від косметичних змін до повних конверсій, які додають нові механіки, нації та сценарії.

По-друге, розробники Paradox написали високоякісну документацію для створення модів, яка за своєю деталізацією не поступається професійним комерційним продуктам, таким як Shopify API. Ця документація ретельно пояснює всі аспекти роботи з ігровим рушієм, структурами файлів та механіками, що робить процес модифікації доступним як для новачків, так і для досвідчених розробників. Окрім того, Paradox активно підтримує спільноту, створюючи навчальні матеріали, відеоуроки та форуми, де можна отримати консультацію. Це значно знижує бар’єр для входу в моддерську діяльність і дозволяє новачкам швидко освоїти інструменти.

Ще одна важлива особливість – моди на ігри Paradox за своєю суттю є неофіційними DLC. Вони використовують ту ж файлову структуру, аналогічні підходи та API для взаємодії з ігровим рушієм, що й офіційні доповнення. Це дозволяє моддерам змінювати гру майже повністю, додаючи складні механіки та великий обсяг контенту, який може сягати гігабайтів. Завдяки цьому, спільнота моддерів може створювати проєкти, які за масштабами й функціональністю мало чим поступаються офіційним DLC, що зберігає інтерес гравців і робить гру актуальною протягом тривалого часу.

Локалізація модифікацій. Як було вже зазначено, кількість фанатського контенту (модів, фанфіків, відео, артів, обговорень) для лише однієї гри Paradox Interactive. Наприклад, Hearts of Iron IV, вражає – понад 130 тисяч модифікацій. Якщо врахувати всі 4X ігри цієї компанії, загальна кількість фанатських

модифікацій перевищує 250 тисяч. Такий обсяг контенту створює величезну потребу у локалізації, адже значна частина гравців хоче мати доступ до модифікацій рідною мовою.

Розуміючи це, спільнота вже почала створювати власні інструменти для локалізації модів. Одним із таких інструментів є ModTranslationHelper – додаток, написаний членом спільноти для спрощення процесу перекладу модифікацій. Цей додаток надає зручний інтерфейс та базові функції для автоматизації перекладів, що значно полегшує життя моддерам і перекладачам (див. рис. 1.9).

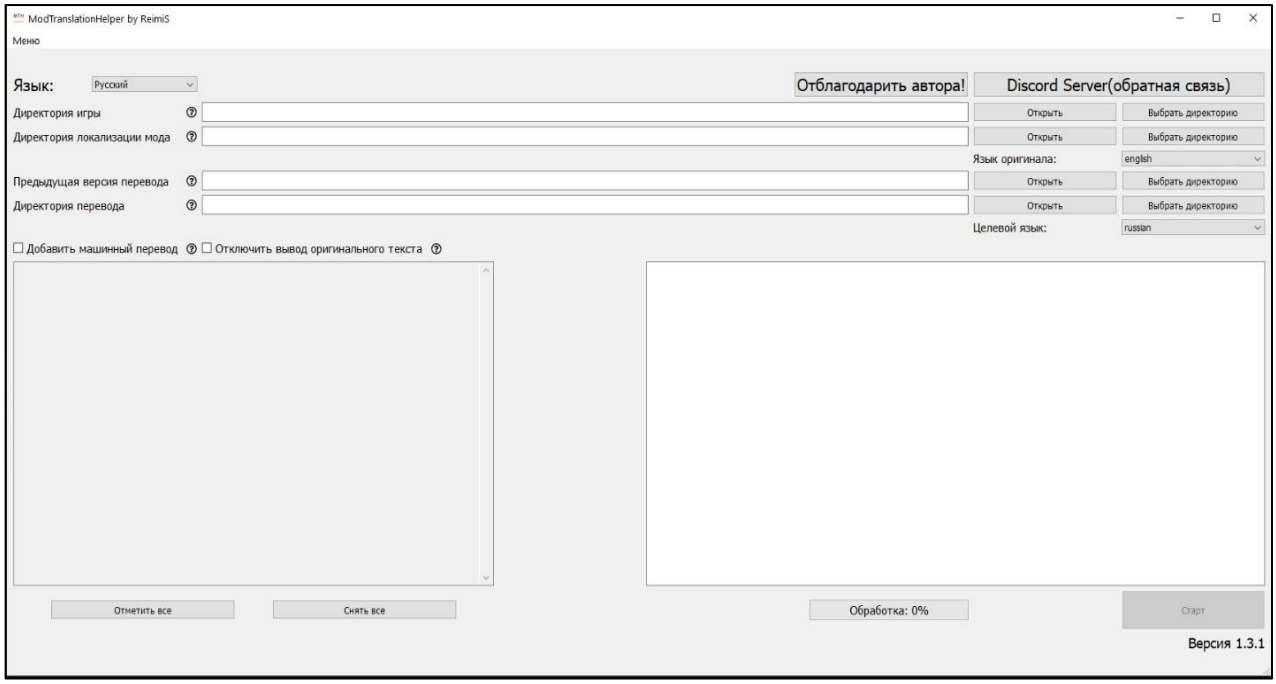


Рисунок 1.9 – UI ModTranslationHelper

Проте ModTranslationHelper має суттєві недоліки. Найбільший з них – це формат додатку. Це окремий додаток для Windows, написаний на C#, який потрібно завантажити та встановити на власний комп’ютер. Він вимагає доступу до інтернету для роботи з API перекладачів, що додає ще одну потенційну точку вразливості. У сучасному цифровому світі, в будь-якому завантаженому файлі може бути прихований шкідливий код, як-от майнери крипто валюти чи інше шкідливе ПЗ, встановлення таких програм є ризикованим. Окрім того, наявність додатків, що працюють лише на одній платформі (Windows), значно обмежує коло користувачів, які можуть скористатися цим інструментом.

Ці недоліки вказують на потребу в більш безпечних та універсальних рішеннях, які не вимагатимуть встановлення програм на комп’ютер користувача, а працюватимуть безпосередньо через браузер. Вебдодатки, які використовують сучасні API для перекладу, можуть забезпечити той самий функціонал без ризиків, пов’язаних із встановленням програмного забезпечення на власний ПК, що робить їх значно привабливішими для користувачів.

Іншим подібним інструментом є Paradox Mod Language Converter, написаний на МП JavaScript з використанням фреймворку Electron. Як і попередній додаток, цей інструмент також має суттєві недоліки, які обмежують його зручність і доступність для користувачів. Основною проблемою залишається той факт, що для використання додатку його необхідно завантажити та встановити на свій комп’ютер. Однак на цьому складнощі не закінчуються – перед використанням додаток потрібно ще й зібрати самостійно.

Для цього користувачеві необхідно мати базові технічні навички: встановити Node.js, виконати команди npm для збірки додатку, і лише потім почати користуватися програмою. Це не тільки додає значну кількість кроків до процесу, але й створює додатковий бар’єр для тих користувачів, які не мають досвіду роботи з інструментами розробки. Такий підхід робить додаток недоступним для більшості гравців, які шукають прості та швидкі рішення для перекладу модів.

Додатково, Paradox Mod Language Converter має ще одну специфічну проблему, пов’язану з API, яке він використовує. Додаток покладається на «Яндекс» API для перекладу, що робить його недоступним для користувачів з України через обмеження на використання сервісів Яндекса у цій країні. Це значно звужує аудиторію користувачів і створює додаткові перешкоди для локалізації модів у великих регіонах, де такі обмеження діють.

Таким чином, попри те, що Paradox Mod Language Converter пропонує функціонал для перекладу модифікацій (див. рис. 1.10). Така технічна складність та використання обмеженого API зменшує його привабливість і доступність для широкої аудиторії. Це ще раз підкреслює необхідність створення вебдодатки, які

будуть простими у використанні, не потребуватимуть встановлення та складної технічної підготовки, а також будуть доступними для користувачів у всьому світі без обмежень на рівні API або платформи.

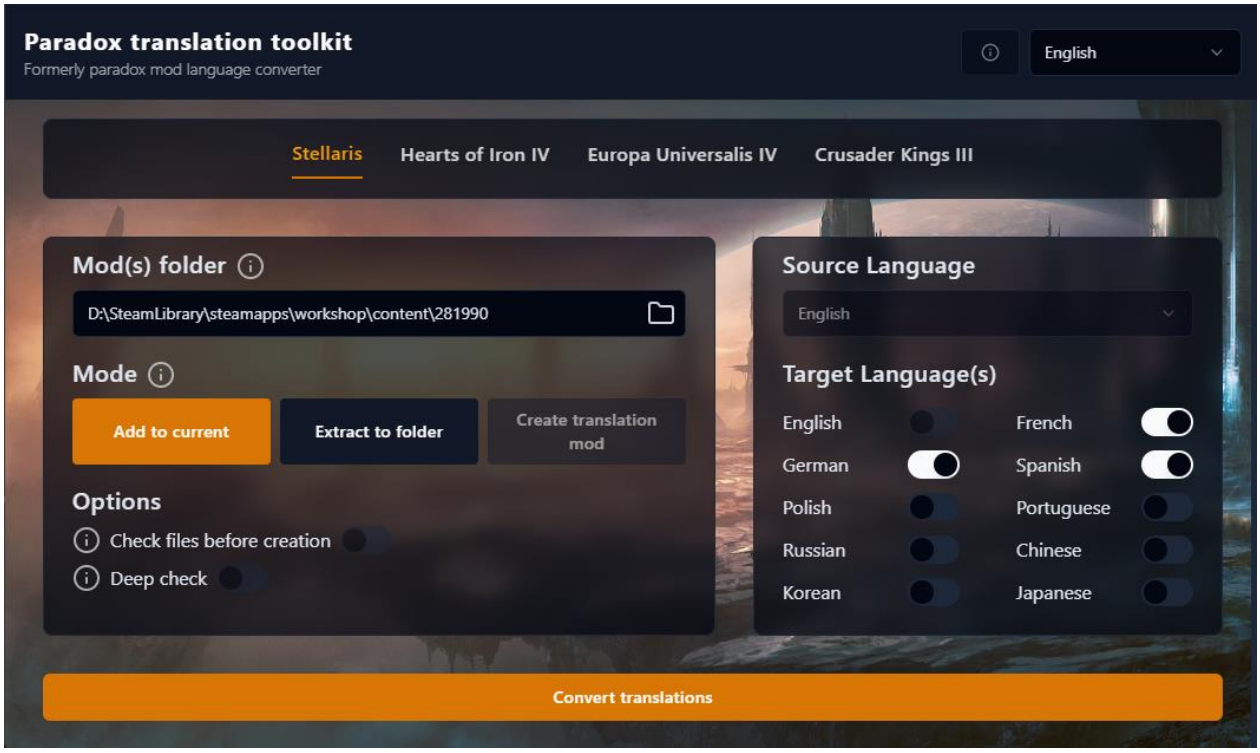


Рисунок 1.10 – UI Paradox Mod Language Converter

1.3 Обґрунтований вибір технологій

У цьому пункті буде розглянуто основні технології, які стали фундаментом для розробки проєкту, а також загальні принципи логіки роботи додатку. Кожна з вибраних технологій відповідає певним вимогам до продуктивності, безпеки та масштабованості, що дозволяє досягти високої ефективності та зручності у використанні. JavaScript (TypeScript): основна мова програмування, яка використовується як для фронтенд, так і для бекенд частин проєкту. TypeScript, що є надбудовою над JavaScript, забезпечує статичну типізацію, підвищуючи надійність та підтримку коду. Node.js: середовище виконання JavaScript на сервері, яке дозволяє ефективно обробляти асинхронні запити та швидко розгортати серверні рішення. Воно є ключовою частиною бекенду проєкту.

Next.js: фреймворк для React, який надає потужні інструменти для серверного рендерингу, покращуючи SEO та продуктивність вебдодатку. Він також дозволяє легко інтегрувати різні API та забезпечує гнучкість у розробці інтерфейсу. Redis: База даних у пам’яті, що використовується для керування чергами запитів. Redis забезпечує високу швидкість і ефективність, що особливо важливо при обробці великої кількості запитів до перекладацьких API. Мікросервісна архітектура: підхід до розробки, який розділяє додаток на невеликі автономні сервіси. Це дозволяє легко масштабувати та оновлювати окремі компоненти системи без впливу на інші. Microsoft Translator API: API для перекладу текстів, яке надає високоякісні переклади з підтримкою великої кількості мов (зокрема, через хмарні сервіси Azure). Воно використовується для автоматизації процесу перекладу модифікацій, забезпечуючи швидке перетворення масивів текстових рядків та дозволяючи перекладачам (людям) зосередитися на тонкому редагуванні, контексті та термінологічній точності.

OpenAI API: API, що дозволяє використовувати потужні моделі штучного інтелекту для генерації або редагування тексту. Використовується для покращення якості перекладу та адаптації тексту. AWS (Amazon Web Services): Хмарна платформа для хостингу, яка забезпечує високу надійність, масштабованість і безпеку для розміщення вебдодатку та обробки запитів.

1.3.1 JavaScript (TypeScript)

JavaScript – це основна мова програмування для веброзробки, яка використовується як на стороні клієнта (фронтенд), так і на серверній стороні (бекенд) за допомогою Node.js. У нашому проєкті використовується його надбудова – TypeScript, яка додає статичну типізацію, що дозволяє запобігати багатьом помилкам на етапі компіляції [11]. Це робить код більш надійним і легшим для підтримки, особливо в масштабних проєктах. TypeScript є особливо корисним, коли потрібно працювати з великими та складними системами,

оскільки він допомагає структурувати код і спрощує процес розробки, зменшуючи кількість потенційних помилок. За рахунок використання типів, TypeScript дає змогу чітко визначати, які дані очікуються в кожному місці, підвищуючи передбачуваність поведінки додатку. Приклад: typeScript коду (див. рис. 1.11).

```
// Оголошення інтерфейсу для запитів перекладу
interface TranslationRequest {
  sourceText: string;
  sourceLanguage: string;
  targetLanguage: string;
}
// Функція, яка приймає об’єкт із типом TranslationRequest та повертає перекладений текст
async function translateText(request: TranslationRequest): Promise<string> {
  const { sourceText, sourceLanguage, targetLanguage } = request;
  // Логіка інтеграції з API перекладу (наприклад, DeepL)
  const translatedText = await deepLApiTranslate(sourceText, sourceLanguage, targetLanguage);
  return translatedText;
}
// Виклик функції
const request: TranslationRequest = {
  sourceText: "Hello, world!",
  sourceLanguage: "en",
  targetLanguage: "de"
};
translateText(request).then(translated => {
  console.log(`Translated text: ${translated}`);
});
```

Рисунок 1.11 – Типовий приклад коду TypeScript

Опис та роз’яснення коду за рядками:

– інтерфейс TranslationRequest: визначає тип об’єкта, який буде передано у функцію для перекладу – це дозволяє чітко визначити структуру запиту;
– функція translateText: приймає об’єкт з параметрами, типізованими як TranslationRequest, виконує асинхронний запит до API перекладу

(умовно через функцію deepLApiTranslate) і повертає перекладений текст;

– виклик функції демонструє використання однієї оголошеної з реальними даними та виводить результат перекладу в консоль [4].

1.3.2 Середовище Node.js

Середовище виконання JavaScript, яке дозволяє запускати код на серверній стороні. Воно побудоване на рушії V8 від Google, що забезпечує швидку обробку JavaScript-коду поза браузером. Node.js активно використовується в сучасній веброзробці для створення серверних додатків, API, мікросервісів, а також для обробки великої кількості асинхронних запитів [22].

Однією з головних особливостей Node.js є подієво-орієнтована, неблокуюча архітектура. Це означає, що він ефективно обробляє безліч одночасних запитів за допомогою асинхронного введення/виведення, що дозволяє досягати високої продуктивності при обробці таких завдань, як HTTP-запити, робота з базами даних, або інтеграція з зовнішніми API. Такий підхід робить Node.js ідеальним вибором для додатків, де критично важливі швидкість та асинхронність [1].

Проте у Node.js є певні недоліки. Один із головних – це низька оптимізація для багатопотокових задач. Оскільки Node.js працює на основі однопотокової моделі, він може бути не найкращим вибором для обробки інтенсивних обчислень або завдань, що вимагають великої кількості ресурсів CPU. Така архітектура добре підходить для роботи з великою кількістю одночасних запитів.

1.3.3 Redis (Remote Dictionary Server)

База даних Redis, яка зберігає дані в оперативній пам’яті, що робить її надзвичайно швидкою для операцій читання та запису. Основною особливістю

Redis є її ключ-значення архітектура, яка дозволяє працювати з даними за допомогою простих операцій. Крім того, Redis підтримує складніші структури даних, такі як списки, множини, хеші та упорядковані множини, що робить її дуже гнучкою для різних випадків використання.

Redis надзвичайно важлива для мікросервісної архітектури завдяки її здатності швидко і ефективно обробляти великі обсяги даних у реальному часі. У нашому проєкті Redis використовується для керування чергами запитів до перекладацьких API, що є критичним для забезпечення стабільної роботи системи з великою кількістю одночасних запитів.

Відсутність необхідності зберігати дані на диску забезпечує низьку затримку при обробці даних, що ідеально підходить для сценаріїв, де потрібна швидка реакція на зміни.

Для роботи з чергами у Redis в середовищі Node.js використовується бібліотека bull. Bull забезпечує простий і потужний інструмент для роботи з чергами завдань, які можуть бути асинхронно оброблені. Основні переваги використання bull:

- просте управління чергами: бібліотека дозволяє легко створювати черги завдань і розподіляти їх між воркерами;
- розширені функції моніторингу: bull надає можливість відстежувати статус кожного завдання в черзі (обробка, успішне виконання, помилка);
- масштабованість завдяки redis, bull забезпечує можливість горизонтального масштабування, що дозволяє додавати нові воркери для розподілу навантаження в реальному часі (приклад використання bull з redis у Node.js наведено на рисунку 1.12.

Прокоментуємо деякі рядки коду. Queue: створюється черга для завдань перекладу. Redis використовується як сервер для зберігання черги. Add(): додає завдання до черги, передаючи дані для перекладу. Process(): обробляє завдання з черги, виконуючи функцію перекладу через зовнішній API. On('completed'): виводить повідомлення про успішне завершення завдання. On('failed'): обробляє помилки, якщо завдання не вдалося виконати [23, 24].

```
import Queue from 'bull';
// Створення черги для запитів до перекладацького API
const translationQueue = new Queue('translation', {
  redis: {
    host: '127.0.0.1',
    port: 6379,
  },
});
// Додавання завдання до черги
translationQueue.add({
  sourceText: 'Hello, world!',
  sourceLanguage: 'en',
  targetLanguage: 'es',
});
// Обробка черги
translationQueue.process(async (job) => {
  const { sourceText, sourceLanguage, targetLanguage } = job.data;
  // Логіка перекладу за допомогою API
  const translatedText = await translateText(sourceText, sourceLanguage, targetLanguage);
  return translatedText;
```

Рисунок 1.12 – Лістинг коду звертання до БД Redis

Таким чином, Redis є критичним компонентом для керування асинхронними завданнями в проєкті, а бібліотека bull дозволяє легко організувати ефективне управління чергами. Це забезпечує стабільну і швидку роботу додатку навіть при високому навантаженні.

1.3.4 Мікросервісна архітектура

Мікросервісна архітектура – це підхід до проєктування програмного забезпечення, при якому додаток розбивається на набір невеликих, незалежних сервісів, кожен з яких виконує певну бізнес-функцію. Кожен мікросервіс працює автономно і може бути розроблений, розгорнутий і масштабований незалежно від інших сервісів. Ці мікросервіси спілкуються між собою через чітко визначені інтерфейси, наприклад, за допомогою HTTP API або системи повідомлень, як Redis чи RabbitMQ. Такий підхід дозволяє значно підвищити гнучкість і продуктивність системи.

Важливими перевагами мікросервісної архітектури є:

– масштабованість: кожен мікросервіс може масштабуватися незалежно від інших, що дозволяє краще використовувати ресурси;
– гнучкість у розробці: команди можуть працювати над різними мікросервісами паралельно, використовуючи різні технології та інструменти;
– зниження ризиків: оскільки мікросервіси працюють незалежно, проблеми в одному з них не призводять до повного краху системи.

У нашому проєкті, що займається перекладом модифікацій для ігор, мікросервісна архітектура є надзвичайно важливою через великий обсяг запитів. Додаток обробляє сотні запитів на переклад рядків з файлів локалізації, що може створити велике навантаження на систему. Завдяки мікросервісам можна розподілити, це навантаження між різними сервісами, кожен з яких спеціалізується на певній задачі. Це дозволяє обробляти запити паралельно, прискорюючи виконання. Легко додавати нові сервіси або масштабувати існуючі для роботи з більшими обсягами даних.

Забезпечити стабільність системи, навіть при великому навантаженні, оскільки кожен мікросервіс працює незалежно. Код нижче наведено спрощений, як приклад, вигляд системи мікросервісів для обробки запитів на переклад за допомогою Redis та мікросервісної архітектури (див. рис. 1.13).

Опис коду:

– translationQueue: використовується для створення черги запитів на переклад (Redis забезпечує швидке зберігання та обробку черги);
– translateService – це окремий мікросервіс, який займається безпосередньою взаємодією з API перекладу, отримуючи перекладений текст;
– process: використовується для обробки завдань з черги. Кожен запит на переклад обробляється незалежно, що забезпечує асинхронність і високу продуктивність;
– on('completed'): даний метод дозволяє відстежувати завершення перекладу і обробляти результат.

```
// Мікросервіс для отримання та обробки запитів перекладу
import Queue from 'bull';
// Створюємо чергу для перекладу
const translationQueue = new Queue('translation', {
  redis: { host: '127.0.0.1', port: 6379 },
});
// Мікросервіс перекладу
async function translateService(text: string, sourceLang: string, targetLang: string) {
  // Логіка взаємодії з API перекладу (наприклад, DeepL)
  const translatedText = await deepLApiTranslate(text, sourceLang, targetLang);
  return translatedText;
}
// Процес обробки черги перекладу
translationQueue.process(async (job) => {
  const { text, sourceLang, targetLang } = job.data;
  const translatedText = await translateService(text, sourceLang, targetLang);
  return translatedText;
});
// Інший мікросервіс для роботи з результатами
translationQueue.on('completed', (job, result) => {
  console.log(`Перекладено: ${result}`);
  // Додаткова логіка обробки результату
});
```

Рисунок 1.13 – Лістинг коду системи мікросервісів для обробки запитів

Такий підхід дозволяє масштабувати систему відповідно до потреб і обробляти великий обсяг даних, розподіляючи навантаження між окремими мікросервісами. Це не тільки покращує продуктивність, але й робить систему більш стійкою до можливих збоїв.

1.3.5 DeepL API

DeepL API – це потужний сервіс машинного перекладу, який надає можливість автоматизованого перекладу текстів із високою якістю. Сервіс був запущений компанією DeepL у 2017 році, за короткий час завоював репутацію найбільш точного та якісного рішення для перекладу текстів серед наявних у світі альтернатив. DeepL надає API, яке дозволяє розробникам інтегрувати можливості перекладу безпосередньо в свої додатки [10].

На сьогоднішній день DeepL визнано найкращим сервісом для перекладу завдяки своїй здатності обробляти контекст і перекладати тексти більш природно, порівняно з іншими аналогами, такими як Google Translate або Microsoft Translator. DeepL використовує нейронні мережі для аналізу великих обсягів тексту, що дозволяє йому генерувати більш точні та контекстно правильні переклади.

Основні особливості DeepL API [25]:

– висока якість перекладу: завдяки використанню передових технологій штучного інтелекту та глибокого навчання, DeepL забезпечує найкращу якість перекладу, особливо у складних мовних парах, таких як англійська-німецька або англійська-французька;
– контекстний переклад: на відміну від багатьох інших сервісів перекладу, які дослівно перекладають окремі слова або фрази, DeepL враховує контекст і значення, що дозволяє уникнути частої проблеми неточних чи двозначних перекладів;
– широка підтримка мов: DeepL підтримує понад 30 мов і регулярно оновлюється для покращення якості та додавання нових мовних пар;
– інтеграція через API: DeepL надає зручне REST API, що дозволяє легко інтегрувати сервіс перекладу у вебдодатки або мобільні додатки, автоматизуючи процес перекладу великого обсягу текстів;
– простий у використанні: API DeepL дозволяє легко надсилати запити на переклад з мінімальними налаштуваннями та швидко отримувати результат, що робить його ідеальним для інтеграції у додатки, які потребують перекладу текстів у реальному часі.

API DeepL працює через HTTPS-запити. Треба надіслати текст для перекладу разом із параметрами вихідної та цільової мов, і отримати відповідь із перекладом. DeepL API на сьогодні є лідером у сфері автоматизованого перекладу текстів завдяки своїй високій точності, контекстному перекладу та здатності працювати з великою кількістю мов. Це робить його ідеальним вибором для проєктів, де критично важлива якість перекладу, як у нашому додатку, де обробляються сотні запитів на переклад текстових модифікацій.

1.3.6 OpenAI API

OpenAI API – це потужний інструмент на базі моделей штучного інтелекту, таких як GPT (Generative Pre-trained Transformer), який використовується для виконання широкого спектра завдань, включаючи генерацію, корекцію та покращення тексту. Завдяки своїм можливостям обробляти текст із високою контекстною точністю та зберігати логічність і зв'язність, OpenAI API стає незамінним інструментом для пост-редактури перекладів та інших текстових операцій [13].

Основні переваги OpenAI API для пост-редактури (див. рис. 1.14):

– контекстна обробка: OpenAI використовує великі обсяги текстових даних для навчання своїх моделей, що дозволяє йому краще розуміти контекст і граматичні нюанси – це критично важливо для пост-редактури, оскільки машинні переклади можуть бути буквальними і потребувати додаткової адаптації під контекст;
– покращення стилю: OpenAI може покращити стиль перекладу, роблячи текст більш плавним, зв'язним і природним для читання – це важливо при перекладі художніх текстів або складних документів, де важлива не тільки точність, але й стилістична відповідність;
– корекція граматичних та синтаксичних помилок: під час машинного перекладу можуть виникати незначні граматичні або синтаксичні помилки. OpenAI API здатен виправляти такі помилки та оптимізувати структуру речень;
– адаптація під специфічну аудиторію: OpenAI API може редагувати текст, беручи до уваги певні стилістичні або культурні особливості, що дозволяє адаптувати переклад під конкретну аудиторію.

Розглянемо роботу OpenAI:

– контекстний підхід OpenAI API використовує модель "text-davinci-003" для обробки тексту та покращення його якості (сервіс отримує перекладений текст і редагує його з метою покращення природності, стилю та ясності);

– temperature параметр визначає креативність відповіді (наприклад: 0.7 дозволяє отримати більш варіативні та менш шаблонні відповіді, що корисно для пост-редактури);
– гнучкість, OpenAI може адаптувати текст під різні вимоги, наприклад, зміна тону, стилю, або навіть спрощення складних речень для кращого розуміння [21].

```
const { Configuration, OpenAIApi } = require('openai');

const configuration = new Configuration({
  apiKey: 'ваш ключ API',
});
const openai = new OpenAIApi(configuration);

async function postEditTranslation(text) {
  try {
    const response = await openai.createCompletion({
      model: 'text-davinci-003',
      prompt: `Покращ переклад наступного тексту для більшої природності і
чіткості: "${text}"`,
      temperature: 0.7,
      max_tokens: 1000,
    });
    return response.data.choices[0].text.trim();
  } catch (error) {
    console.error('Помилка під час пост-редактури:', error);
  }
}

// Використання функції для пост-редактури
const translatedText = 'The book is very good and interesting for reading.';
postEditTranslation(translatedText).then((editedText) => {
  console.log('Відредагований текст:', editedText);
});
```

Рисунок 1.14 – Лістинг коду використання OpenAI API для пост-редактури перекладу

Ми вважаємо, що OpenAI – найкращий варіант для пост-редактури, тому що він здатний аналізувати складні структури речень, граматичні конструкції та навіть стилістичні особливості, що робить його ідеальним інструментом для тонкої роботи з текстом. Після машинного перекладу текст може втрачати логічну зв’язність або плавність. OpenAI може відновити ці елементи, створюючи текст, який легко зчитається і виглядає як робота професійного перекладача. OpenAI API дозволяє обробляти великі обсяги текстів, що є ключовим для додатків, які працюють з перекладом у реальному часі або обробляють масиви даних.

OpenAI API – це найкращий варіант для пост-редактури перекладів у нашому додатку, оскільки він дозволяє автоматично покращувати якість текстів, виправляти помилки, адаптувати стиль та підвищувати природність перекладу. Це робить наш підхід до перекладу більш ефективним і точним, що особливо важливо при роботі з великою кількістю запитів на переклад у реальному часі. Завдяки гнучкості моделі можна налаштовувати тон та стиль тексту відповідно до вимог конкретної модифікації чи мовної групи. Крім того, API легко інтегрується у мікросервісну архітектуру, забезпечуючи стабільну й масштабовану роботу сервісу.

1.4 Висновки до першого розділу

У першому розділі кваліфікаційного проєкту було проведено детальний аналіз особливостей ігор Paradox Interactive, зокрема їхнього жанру, механік та впливу на стабільність аудиторії. Порівняння статистики онлайну для різних ігор показало явну залежність стійкості та зростання аудиторії від обсягу фанатського контенту, який створюється спільнотою. Ігри Paradox, такі як Hearts of Iron IV, демонструють високу продуктивність та активність спільноти моддерів, що суттєво подовжує життєвий цикл гри і підтримує стабільний інтерес гравців протягом багатьох років.

Окремо було досліджено причини такої аномальної продуктивності спільноти моддерів. Виявлено, що значну роль у цьому відіграють надані розробниками інструменти для модифікації, висока якість документації та активна підтримка навчальними матеріалами. Крім того, моди для ігор Paradox є неофіційними аналогами офіційних DLC, що дає моддерам майже ті ж самі можливості для зміни гри, які мають розробники.

Також було проведено аналіз існуючих інструментів для перекладу модифікацій, таких як ModTranslationHelper та Paradox Mod Language Converter. Незважаючи на їхню корисність, ці додатки мають суттєві недоліки, зокрема

потребу в установці на ПК, обмеження за платформами та технічні складнощі з використанням. Окрім того, деякі інструменти використовують API, які недоступні для користувачів з окремих країн, що обмежує їхню ефективність.

Зокрема, було обґрунтовано визначено основні технології, таких як: JavaScript (TypeScript) для фронтенду та бекенду, Node.js для серверної частини, Redis для керування чергами завдань, а також DeepL API і OpenAI API для автоматичного перекладу та пост-редактури текстів.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

2.1 Архітектура додатку: взаємодія компонентів та потоки даних

Розроблений вебсервіс реалізує мікросервісну архітектуру, у якій кожен функціональний блок виконує окрему задачу як gRPC-інтерфейси використовуються лише для керування процесами (control pl- від обробки запитів користувача до перекладу, постредакції й завантаження результатів. Мікросервіси взаємодіють через Redis Streams, що виступає шиною подій, тоді ane).

Основні компоненти системи:

– web-app – клієнтський інтерфейс на Next.js [26, 27];
– api-service – шлюз REST/gRPC для користувацьких запитів;
– coordinator-service – управління задачами перекладу, постановка у чергу, контроль стану;
– translation-service, postedit-service, special-chars-service, aggregate-service, uploader-service – етапи обробки файлів;
– redis – черга повідомлень та кеш;
– minio – файлове сховище;
– postgresql – реляційна база даних для користувачів та історії перекладів.

Потік даних відбувається за схемою: Ingest → Translate → Postedit, OpenAI → SpecialChars → Aggregate → Upload.

Кожен етап читає повідомлення зі свого стріму Redis, обробляє частину даних і публікує результат у наступний стрім.

Такий підхід забезпечує асинхронність, стійкість до збоїв і горизонтальне масштабування.

2.1.1 JavaScript (TypeScript) та Node.js: переваги для серверної розробки

Серверна частина системи реалізована на Node.js з використанням TypeScript. Таке поєднання було обране свідомо, оскільки воно забезпечує високу швидкість розробки, гнучкість інтеграцій і спрощує підтримку великої кількості мікросервісів, не жертвуючи стабільністю та структурованістю коду.

Node.js базується на неблокувальній моделі введення/виведення (I/O), що робить його оптимальним для систем із великою кількістю одночасних з'єднань і подій.

Завдяки подієвій архітектурі event loop сервер здатен обробляти тисячі запитів паралельно без необхідності створювати новий потік для кожного клієнта, як це робиться у традиційних мовах типу Java чи C# дивитись можна порівняння в таблиці 2.1.

Таблиця 2.1 – Порівняння з іншими мовами (наприклад, C# /.NET)

Критерій	**Node.js / TypeScript**	**C# / .NET**
Модель виконання	Неблокувальна, подієва (event loop)	Потокова, синхронна
Швидкість розробки	Дуже висока, мінімальна кількість шаблонного коду	Середня, потрібні конфігурації, DI-контейнер, налаштування
Продуктивність (CPU)	Нижча в обчисленнях, але добра для I/O	Вища у важких обчисленнях
Простота розгортання	Легка, контейнерні Node-образи ~150 MB	Важчі .NET образи ~ 500 MB
Екосистема бібліотек	Npm – найбільша у світі	NuGet – менша за обсягом
Кросплатформеність	Повна (Windows, Linux, macOS)	Також підтримується, залежить від середовища
Критерій	Node.js / TypeScript	C# / .NET
Використання у фронт/ бекенд	Єдина мова (JS/TS)	Фронт – інші мови (JS, Blazor т.д.)
Швидкість прототипування	Максимальна	Помірна

Основні переваги Node.js та TypeScript:

– єдина мова для фронтенду та бекенду: оскільки клієнтська частина (Next.js) і серверна логіка написані однією мовою – JavaScript/TypeScript – це спрощує розробку, зменшує контекстні перемикання для команди й дозволяє повторно використовувати типи, моделі та утиліти на обох сторонах;
– статична типізація (TypeScript): TypeScript додає статичну перевірку типів і підтримку об’єктно-орієнтованого програмування, що дозволяє уникати більшості типових помилок на етапі компіляції – це поєднує простоту JavaScript із надійністю мов класу C# або Java;
– розвинена екосистема NPM: менеджер пакетів npm є найбільшою екосистемою бібліотек у світі – понад мільйон модулів, що в рази перевищує кількість доступних пакетів у NuGet (C#);
– висока швидкість розробки та зручність інтеграцій: Node.js має низький “time-to-market”: створення нового мікросервісу або endpoint-а займає мінімум часу – у поєднанні з hot-reload, Nest CLI і сучасними інструментами збірки (esbuild, ts-node) розробники можуть зосередитись на логіці, а не на інфраструктурі [15];
– простота мікросервісної інтеграції: у нашій архітектурі основний сервер (Coordinator) реалізовано на NestJS, що забезпечує чітку структуру модулів, контролерів і сервісів, а виконавчі мікросервіси (translate, postedit, special-chars тощо) написані на «чистому» Express.js для максимальної легкості й швидкодії – такий підхід дозволяє поєднати строгість Nest (для orchestration і API) з простотою та легкістю Express (для вузьких воркерів);
– висока підтримуваність коду: TypeScript забезпечує автодоповнення, рефакторинг і чітку документацію через інтерфейси – код стає самодокументованим, що критично для командної розробки мікросервісної системи [3, 8].

Отже, вибір Node.js+ TypeScript у цьому проєкті є усвідомленим рішенням. Ми жертвуємо певною продуктивністю процесора заради читабельності,

гнучкості, величезної екосистеми NPM і надзвичайно швидкої розробки. У контексті мікросервісної архітектури, де кожен сервіс виконує короткі I/O-операції (читання файлу, виклик API, запис у Redis), Node.js показує себе максимально ефективно.

2.1.2 Redis для кешування та черг повідомлень

У даному проєкті Redis виконує роль не лише кешу, а й центральної шини подій для обміну даними між мікросервісами. Це один із ключових елементів архітектури, який забезпечує асинхронність, балансування навантаження, гарантію доставки повідомлень і стійкість до збоїв.

Redis як кеш даних, використовується для тимчасового зберігання даних, доступ до яких має бути максимально швидким:

– сесії користувачів і токени авторизації;
– проміжні результати перекладів (наприклад, чернетки OpenAI);
– короткі відповіді API, які часто повторюються;
– лічильники активних процесів і rate limit.

Операції SET, GET, EXPIRE і INCRBY дозволяють реалізувати просте та ефективне кешування без участі бази даних, що зменшує навантаження на PostgreSQL і пришвидшує обробку запитів.

Redis Streams як система передачі повідомлень. Для комунікації між мікросервісами використовується механізм Redis Streams, який є гібридом черги (queue) і журналу подій (log). Streams дозволяють працювати з повідомленнями у вигляді впорядкованої послідовності записів, кожен з яких має власний ідентифікатор (<timestamp>-<sequence>).

Архітектура потоків у системі перекладу. Кожен етап обробки файлу має власний стрім (lines:in -> lines:translated -> lines:postedited -> lines:cleaned -> file:ready):

– ingestWorker – додає початкові рядки у lines:in;

– translateWorker – зчитує рядки з lines:in, перекладає і публікує у lines:translated;
– rosteditWorker – виконує постредакцію за допомогою OpenAI і публікує в lines:postedited;
– specialCharsWorker – очищує текст і записує у lines:cleaned;
– aggregateWorker – збирає всі оброблені рядки у фінальний файл, зберігає в MinIO і повідомляє систему про завершення.

Таким чином, Redis виступає центральною подійною шиною, через яку дані рухаються між сервісами.

Consumer Groups і механізм XREADGROUP. Redis Streams реалізує модель споживачів за допомогою consumer groups. Кожен мікросервіс створює власну групу, наприклад: XGROUP CREATE lines:translated translate-group $. Далі воркери читають дані командою XREADGROUP GROUP translate-group worker-1 COUNT 10 BLOCK 2000 STREAMS lines:translated, це дозволяє:

– розподіляти навантаження між кількома інстансами одного сервісу;
– уникати дублювання обробки (повідомлення доставляється лише одному споживачу);
– гарантувати збереження непідтверджених повідомлень у разі збою.

Після успішної обробки воркер відправляє команду XACK, що видаляє повідомлення з черги, а при помилках – воно потрапляє у pending list, звідки може бути повторно прочитане (XCLAIM).

Переваги Redis Streams порівняно з RabbitMQ, Kafka чи простими чергами можна переглянути в таблиці 2.2.

Для архітектури, де повідомлення короткі, потік великий, а затримка критична, Redis Streams є найкращим варіантом. Kafka була б надмірною, а RabbitMQ вимагав би додаткової конфігурації і зберігання черг на диску.

Використання Redis у системі Redis виступає одразу в кількох ролях:

– шина подій (Event Bus): кожен сервіс (Worker) підписаний на власний стрім – події рухаються за конвеєром, формуючи повноцінну event-driven архітектуру;

– кеш стану обробки файлу Coordinator-сервіс зберігає у Redis статистику виконання – кількість оброблених рядків, статуси та відсоток прогресу, що дозволяє клієнту отримувати дані миттєво без запиту до бази даних;
– rate limiting і throttling: використання структур типу ZSET або INCR дозволяє лімітувати частоту викликів OpenAI API та запобігати перевищенню квот;
– семафори для координації: для уникнення одночасної обробки рядків одного файлу використовується «віконний семафор», тимчасовий ключ з TTL, який гарантує, що не більше N воркерів працюють над одним файлом;
– dLQ (Dead Letter Queue): повідомлення, які не вдалося обробити після кількох спроб, потрапляють у стрім dlq:<stage> для подальшого аналізу або повторного запуску.

Таблиця 2.2 – Переваги Redis Streams порівняно з RabbitMQ, Kafka

Критерій	**Redis Streams**	**RabbitMQ / Kafka**
Простота налаштування	Мінімальна конфігурація, інтеграція через бібліотеки для Node.js	Потрібні брокери, топіки, ACL
Швидкість обробки	Дуже висока (операції в пам’яті)	Висока, але залежить від дискових логів
Збереження повідомлень	Доступно (RDB/AOF, TTL)	Постійне, але повільніше
Підтримка consumer groups	Є (з 5.0)	Є
Простота для мікросервісів на Node.js	Ідеальна та прості XADD/XREAD/XACK	Потребує зовнішніх клієнтів та брокера
Залежності	Один контейнер Redis	Декілька сервісів (broker, zookeeper, schema-registry)

Ми вважаємо, що Redis ідеальний вибір саме для цього проєкту, адже підтвержується якісна швидкість, працює повністю в пам’яті, що не дає затримки в межах мілісекунд навіть при тисячах повідомлень на секунду. Не потрібен

зовнішній брокер чи спеціальний сервер, лише один контейнер Redis. Підходить як для кешу, так і для потоків подій, а одна технологія для кількох задач (гнучкість). Масштабованість дозволяє горизонтальне масштабування через shard-и і consumer groups. Сумісність із Node.js: клієнти ioredis і redis@v4 мають повну підтримку XADD/XREADGROUP, що спрощує інтеграцію в Express і NestJS. Просте відновлення після збоїв: Redis зберігає стан стрімів, що дозволяє продовжити обробку без втрат навіть після рестарту контейнера.

2.1.3 Масштабованість мікросервісної архітектури

Мікросервісна архітектура забезпечує:

– незалежне масштабування окремих компонентів;
– ізоляцію збоїв (відмова одного сервісу не блокує інші);
– можливість оновлення без зупинки системи (rolling updates);
– використання контейнеризації (Docker, Docker Compose) для розгортання;
– просте розширення – додавання нових етапів обробки без зміни існуючих [9].

Redis Streams – підтримує розподіл навантаження між воркерами через механізм “sharding”, повідомлення різних файлів надходять у різні стріми, а кілька воркерів можуть паралельно споживати події однієї групи [14].

2.2 Структура файлів локалізації Paradox-ігор та їх обробка

Файли локалізації ігор Paradox мають формат .yml, який містить пари “ключ–значення”. Ключ – це, ідентифікатор текстового рядка, а значення – його переклад певною мовою. Перед початком перекладу Coordinator-сервіс розділяє файл на окремі рядки та надсилає їх до Redis Stream lines:in:{shard}. Подальші

сервіси обробляють рядки послідовно, а Aggregate-сервіс об’єднує результати в один фінальний .yml файл, який зберігається в MinIO. Таким чином забезпечується паралельна обробка великих файлів.

2.3 Структура бази даних

База даних проєкту кваліфікаційної роботи реалізована через ORM Prisma, використовує СУБД PostgreSQL.

Складається з трьох основних таблиць:

– таблиця “User”;

– таблиця “Translates”;

– таблиця “Files”.

Розбір даних, які увійшли в перелік, схематично наведено в таблицях за номерами 2.3 – 2.5.

Таблиця 2.3 – Користувач “User”

Поле	**Тип**	**Опис**
id	Int (PK)	Ідентифікатор користувача
email	String (unique)	Електронна адреса
password	String	Хешований пароль
createdAt	DateTime	Дата створення

Таблиця 2.4 – Файли “Files”

Поле	**Тип**	**Опис**
file_id	varchar	Первинний ключ
status	varchar	Поточний стан виконання завдання
total_lines	int	Загальна кількість рядків у файлі
total_chunks	int	Загальна кількість блоків
origin_lang	varchar	Вихідна мова файлу (для конвеєра)
target_lang	varchar	Цільова мова перекладу
created_at	timestamp	Дата і час постановки завдання в чергу
completed_at	timestamp	Дата і час фактичного завершення перекладу
error	varchar	Повідомлення про помилку, якщо status = 'error'

Таблиця 2.5 – Перекладач “Translates”

Поле	**Тип**	**Опис**
id	Int (PK)	Ідентифікатор перекладу
userId	Int (FK)	Посилання на користувача
originalFile	String	Шлях до вихідного файлу
translatedFile	String	Шлях до перекладеного файлу
originLang	String	Мова оригіналу
translatedLang	String	Мова перекладу
createdAt	DateTime	Дата створення запису

Побудовано схему бази даних в db.io (див. рис. 2.1). Даний сервіс використовується для створення діаграм зв’язків баз даних з використанням відповідної мови DSL і підтримує сумісні редагування та генерації.

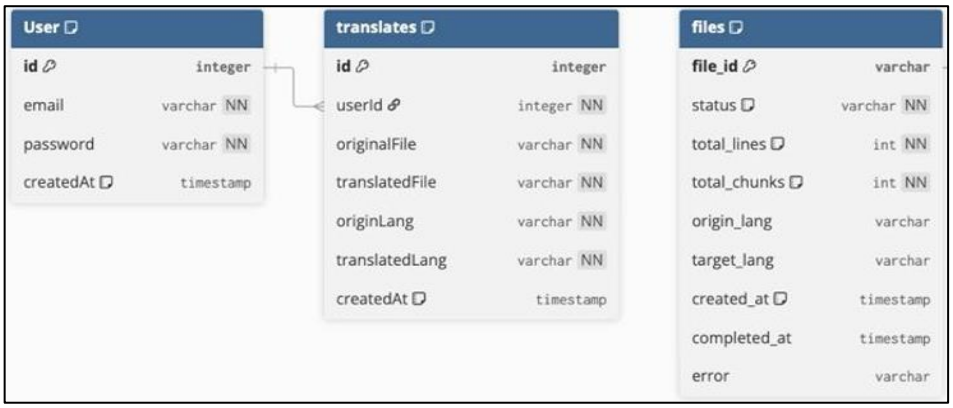


Рисунок 2.1 – Схема бази даних

Вид зв’язка між таблицями «один-до-багатьох»: User 1 – N Translates. Зв’язок «один-до-багатьох» має місце, коли ключовим є тільки одне з полів зв’язку. Таблиця Files зв’язків немає.

2.4 Діаграма прецедентів (Use Case Diagram)

Дана діаграма в UML (Unified Modeling Language), яка візуалізує взаємодію між акторами (людьми чи іншими системами) та прецедентами (випадками використання системи).

Основні сценарії використання системи користувачем (див. рис. 2.1 та 2.2):

– реєстрація та авторизація;
– завантаження файлу локалізації;
– запуск перекладу;
– моніторинг статусу;
– завантаження готового файлу;
– перегляд історії перекладів.

Кожен із зазначених прецедентів відображає ключові функціональні можливості вебсервісу та визначає, як саме користувач взаємодіє із системою на різних етапах робочого процесу. Діаграма прецедентів дозволяє побачити загальну картину роботи сервісу, окреслити межі відповідальності кожного компонента та зрозуміти логіку переходу між діями. Діаграма прецедентів служить фундаментом для подальшого проєктування системи, оскільки саме на її основі формується архітектура додатка, визначаються компоненти бізнес-логіки та встановлюються вимоги до інтерфейсу користувача. Вона забезпечує чітке розуміння того, які задачі має вирішувати система і як вона повинна функціонувати у рамках поставленого завдання [20].

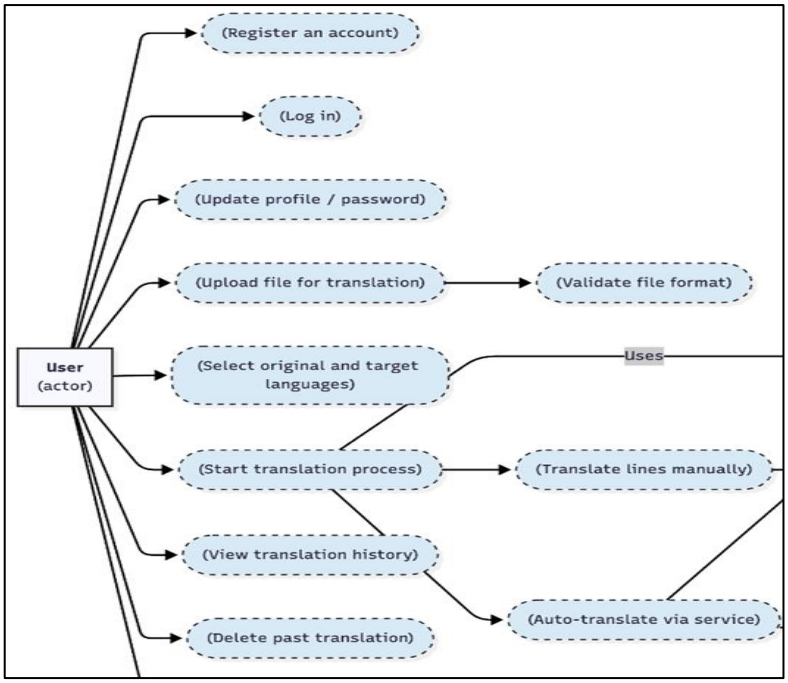


Рисунок 2.2 – Діаграма прецедентів (1 частина)

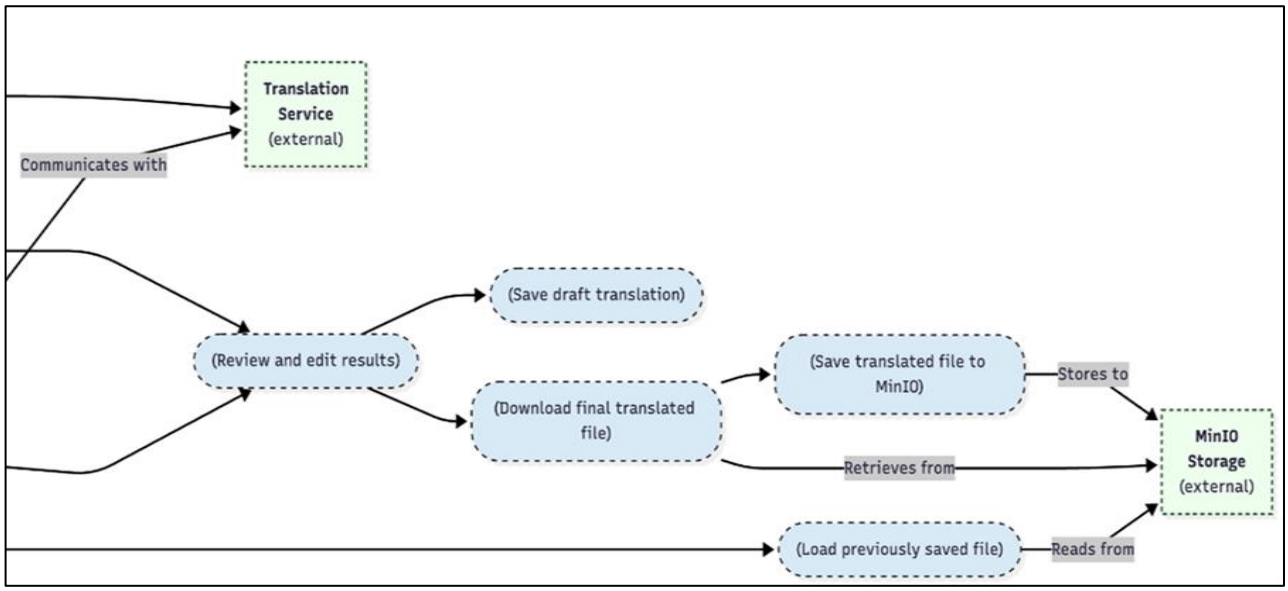


Рисунок 2.3 – Діаграма прецедентів (2 частина)

Отже, діаграма прецедентів є важливим елементом моделювання системи, оскільки дозволяє структуровано відобразити всі ключові сценарії взаємодії користувача з вебсервісом. Вона слугує основою для подальшого проєктування архітектури, визначення функціональних вимог та побудови логічної структури програмного продукту. Завдяки аналізу прецедентів забезпечується цілісне бачення роботи системи, що сприяє ефективній розробці та впровадженню сервісу перекладу модифікацій Paradox Interactive [20].

2.5 Діаграма діяльності (Activity Diagram)

Діаграма діяльності можна сказати є розширеним варіантом блок-схем. Проте блок-схема не має стандартизованої нотації та не містить паралелізмів та синхронізації.

Діаграма діяльності проєкта відображає життєвий цикл обробки файлу, за наступним переліком:

– користувач;

– завантаження;

– постановка у чергу;

– переклад;

– постредакція;

– очищення спецсимволів;

– агрегація;

– збереження результату (див. рис. 2.4 – 2.8).

Дана діаграма нашого проєкту можна сказати є ключовою частиною і представлена уніфікованою мовою моделювання (UML) і використовуються для моделювання бізнес-процесів, деталізації варіантів використання (use cases), а також визначення поведінки операції.

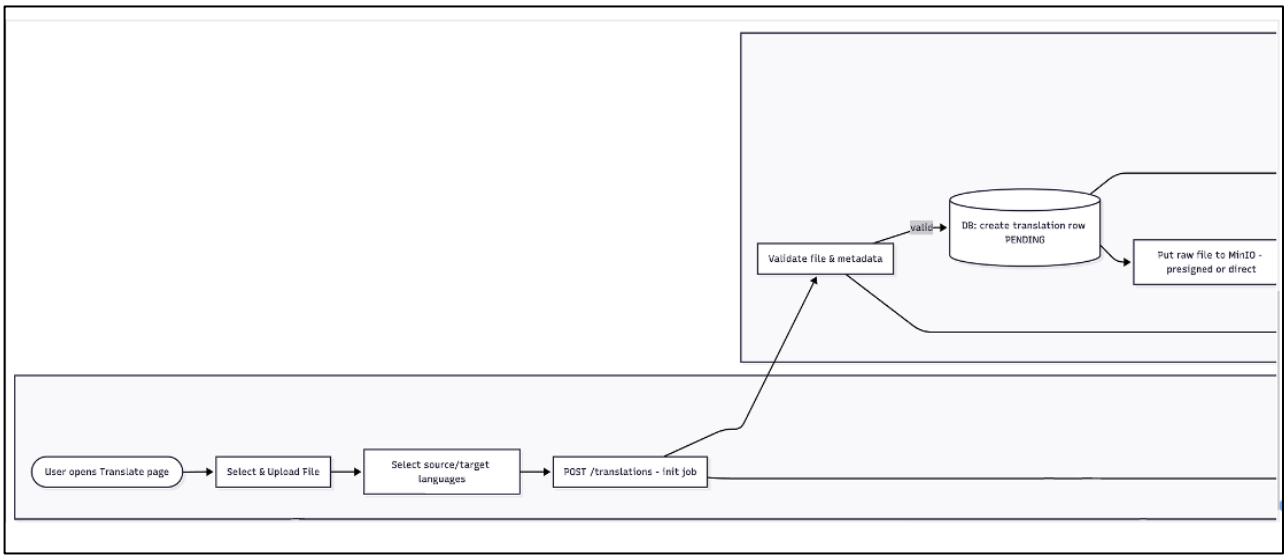


Рисунок 2.4 – Діаграма діяльності три головних дії (1 частина)

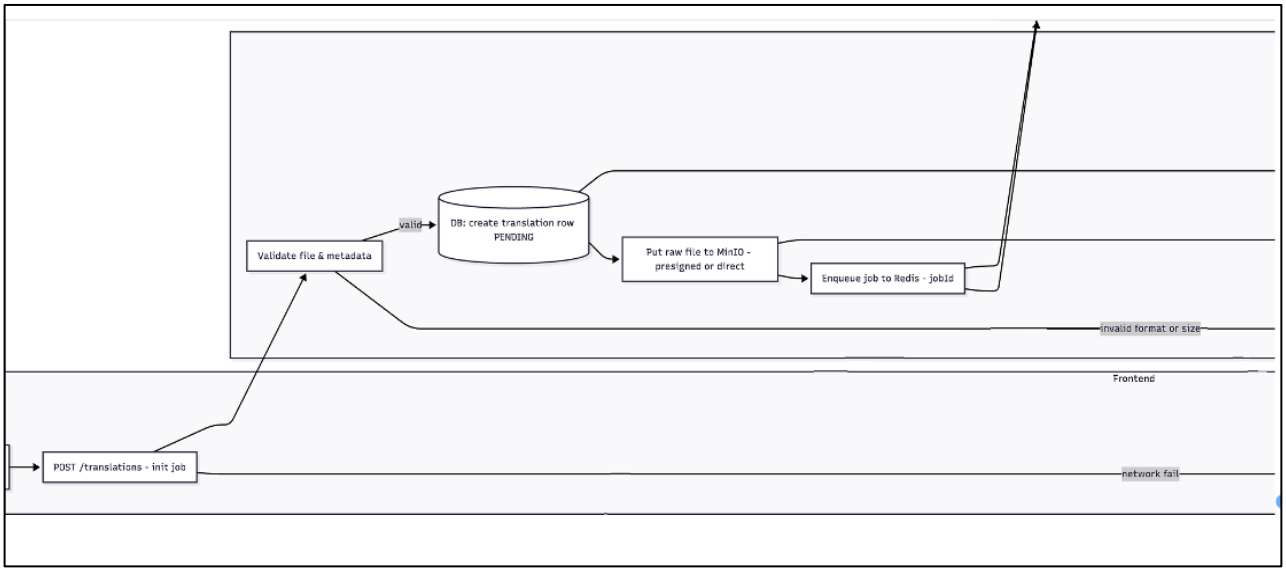


Рисунок 2.5 – Діаграма діяльності три головних дії (2 частина)

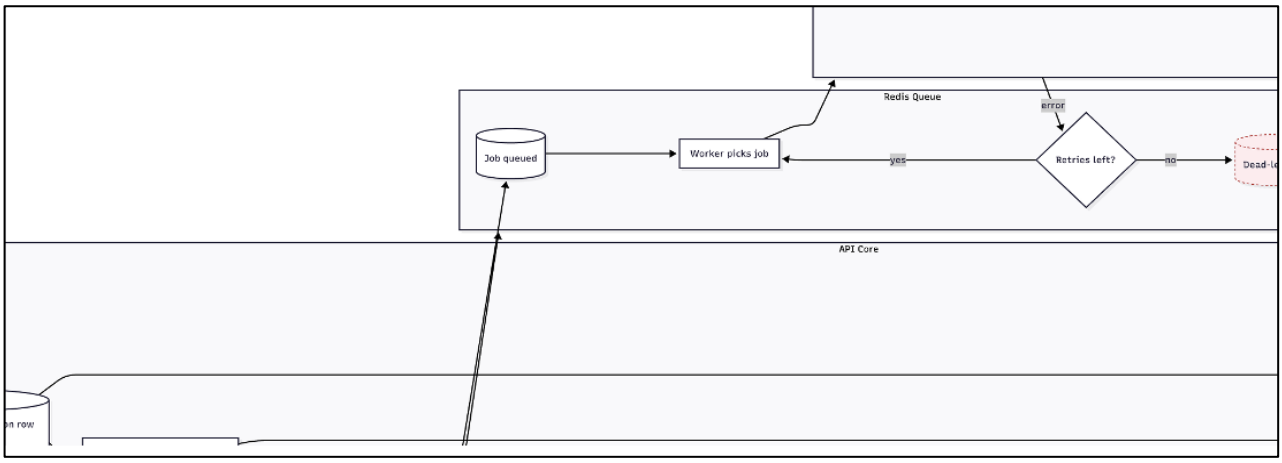


Рисунок 2.6 – Діаграма діяльності три головних дії (3 частина)

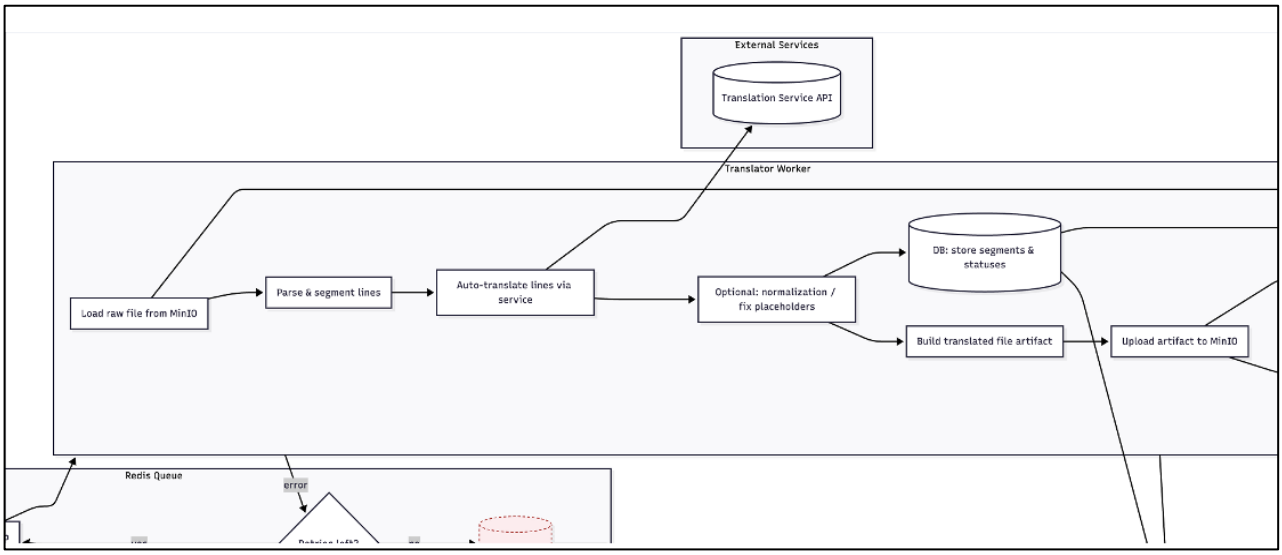


Рисунок 2.7 – Діаграма діяльності три головних дії (4 частина)

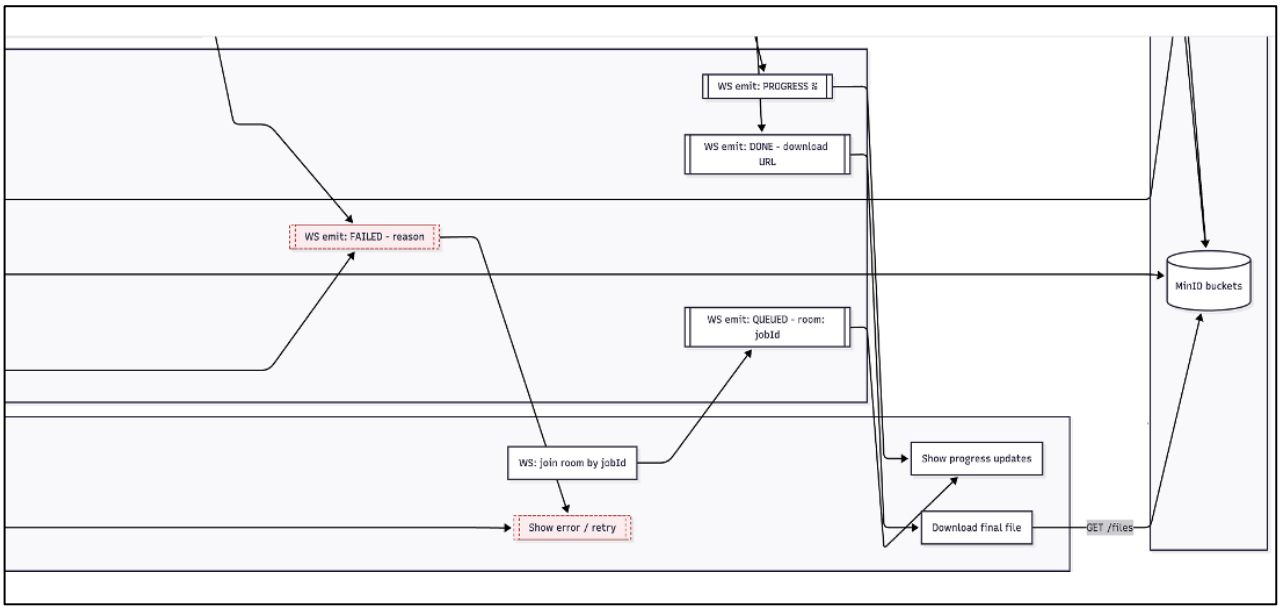


Рисунок 2.8 – Діаграма діяльності три головних дії (5 частина)

2.6 Діаграма компонентів (Component Diagram)

Діаграма компонентів служить частиною фізичного представлення моделі системи та грає важливу роль в процесі об’єктно-орієнтованого аналізу і проєктування, є необхідною для генерації програмного коду. Вона показує різні компоненти системи, залежності між ними та використовує специфічні позначення.

Діаграма компонентів є важливим інструментом для розробки надійних, модульних та легко підтримуваних програмних систем Ключові елементи діаграми можна переглянути в таблиці 2.6.

Таблиця 2.6 – Елементи діаграми компонентів

Елемент	**Опис**
Component	Прямокутник із двома маленькими прямокутниками у верхньому лівому куті. Представляє модульну частину системи.
Інтерфейс постачання	Позначається символом «кульки» і показує послуги, які цей компонент надає іншим компонентам.
Required Interface	Позначається символом «розетки» і показує послуги, які компонент потребує від інших компонентів для своєї роботи.
Dependency	Пунктирна стрілка. Вказує, що один компонент залежить від іншого (наприклад: «Клієнтський інтерфейс залежить від компонента бази даних»).

Створена діаграма нашого проєкту показує взаємодію між мікросервісами: redis, minIO, postgreSQL та зовнішніми користувачами.

Часто компоненти містять або використовують інтерфейси. Якщо взяти діаграми класів, яка фокусується на внутрішній логіці та методах, діаграма компонентів пропонує високорівневий, архітектурний погляд на систем. Можна сказати формулює та показує фізично розвинутий програмний комплекс. Дану діаграму можна представляти постійно в різних гарних проєктах різних направленостей по певним спеціальностям.

Сервіси взаємодіють через шину подій Redis Streams, забезпечуючи асинхронну обробку даних і масштабованість системи (див. рис. 2.9)

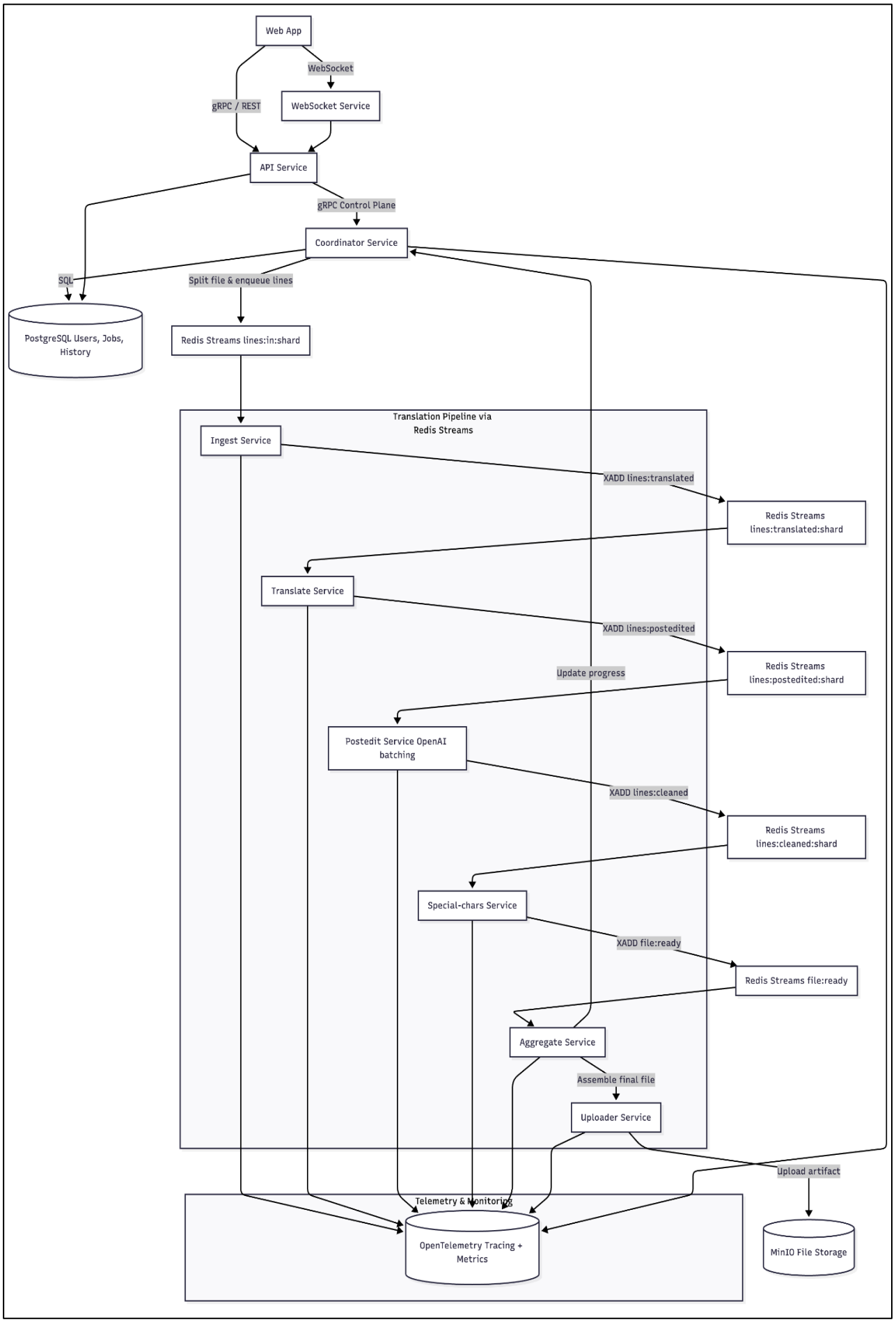


Рисунок 2.9 – Діаграма компонентів

2.7 Діаграма класів (Class Diagram)

Містить модель даних Prisma (User, Translate) і допоміжні класи сервісів (CoordinatorService, WorkerBase, TranslateWorker, PosteditWorker, тощо). Показано зв’язки асоціації, наслідування та композиції між класами. На діаграмі класів представлені такі елементи:

– класи;

– інтерфейси;

– залежності;

– узагальнення;

– асоціації.

Кількість класів складає: 26 елементів та 27 залежних дій (див. рис. 2.10 – 2.17).

Дана діаграма в проєкті являється самою масштабованою діаграмою. Така діаграма використовується для моделювання системи, розбиття її на окремі компоненти та розуміння взаємозв’язків між ними.

Графічно клас зображується як прямокутник із трьома відділеннями для назви, атрибутів та методів відповідностей. Мають прямокутну форму яка ділиться на два відділення. Завжди верхньому відділенні прямокутника, надрукована напівжирним шрифтом і вирівняна по центру. В атрибуті у середньому відділенні вказані змінні класу. Вони вирівняні по лівому краю, перша літера – мала.

В методах (операції) – це нижне відділення де перелічені дії, які може виконувати клас. Вони також вирівняні по лівому краю, перша літера – мала.

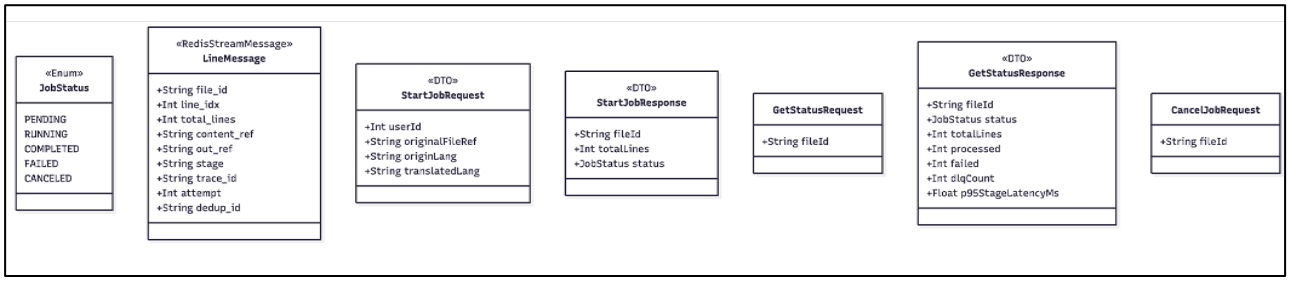


Рисунок 2.10 – Діаграма класів (1 частина)

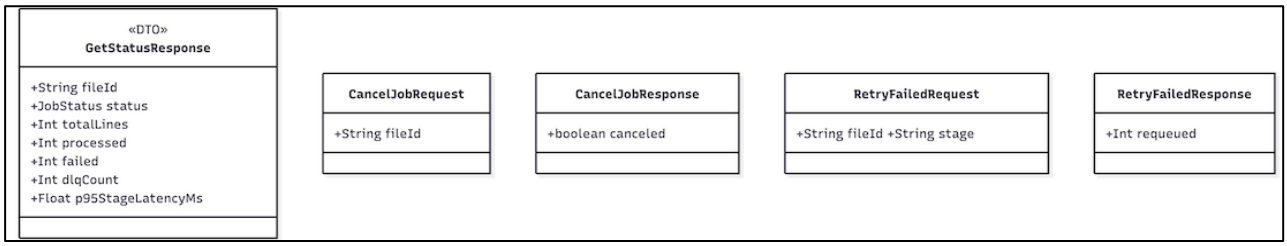


Рисунок 2.11 – Діаграма класів (2 частина)

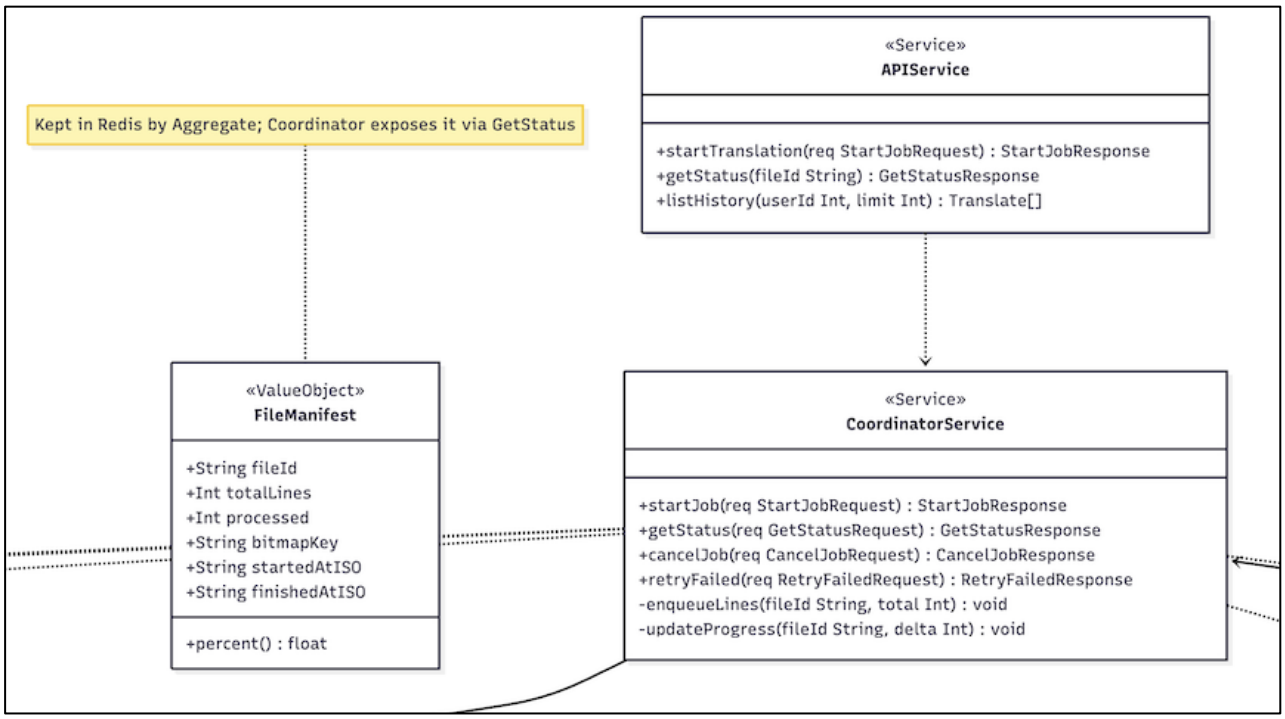


Рисунок 2.12 – Діаграма класів (3 частина)

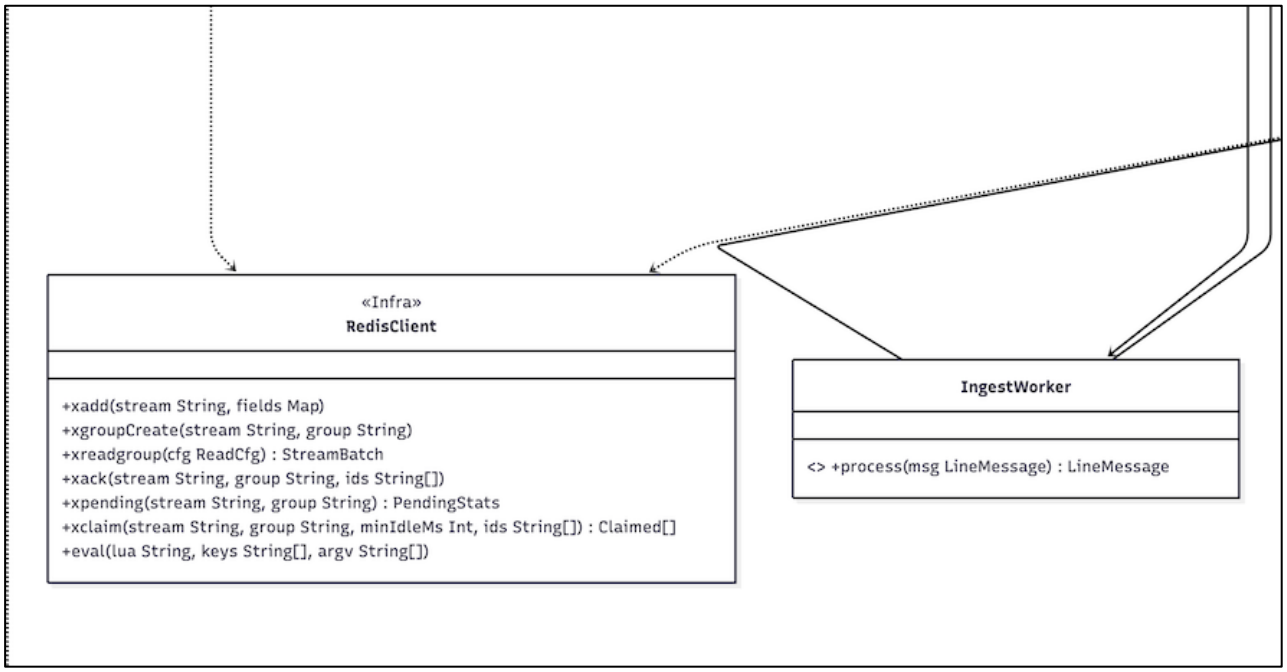


Рисунок 2.13 – Діаграма класів (4 частина)

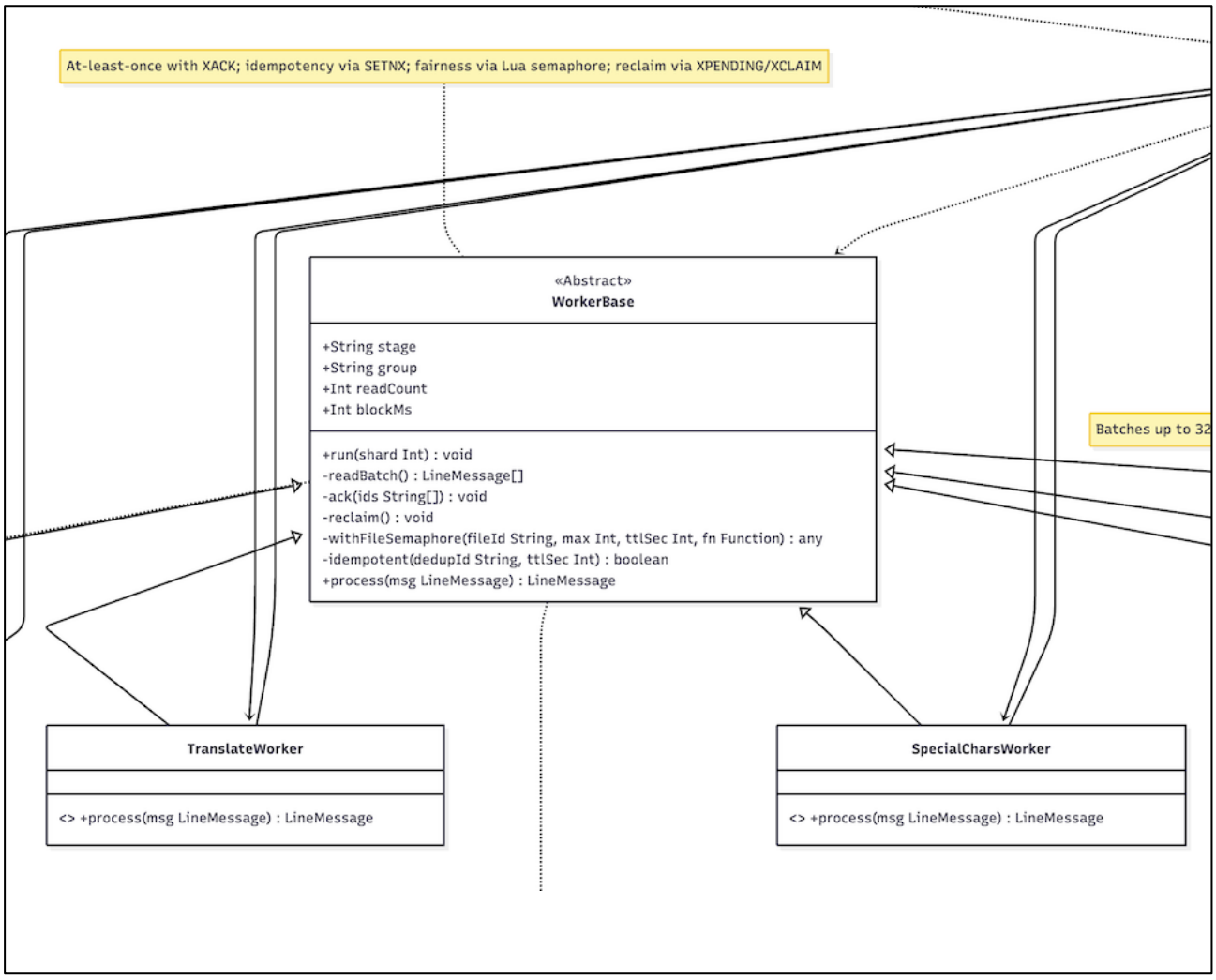


Рисунок 2.14 – Діаграма класів (5 частина)

Зв’язки – це лінії, що з’єднують класи, показують їхні взаємозв’язки, наприклад, асоціації, наслідування або залежності.

Головне призначення таких діаграми:

– документування архітектури програмного забезпечення;
– представлення логічної структури системи;
– відображення статичних елементів системи, таких як класи, інтерфейси та їхні залежності;
– моделювання об’єктно-орієнтованих систем та їхніх компонентів.

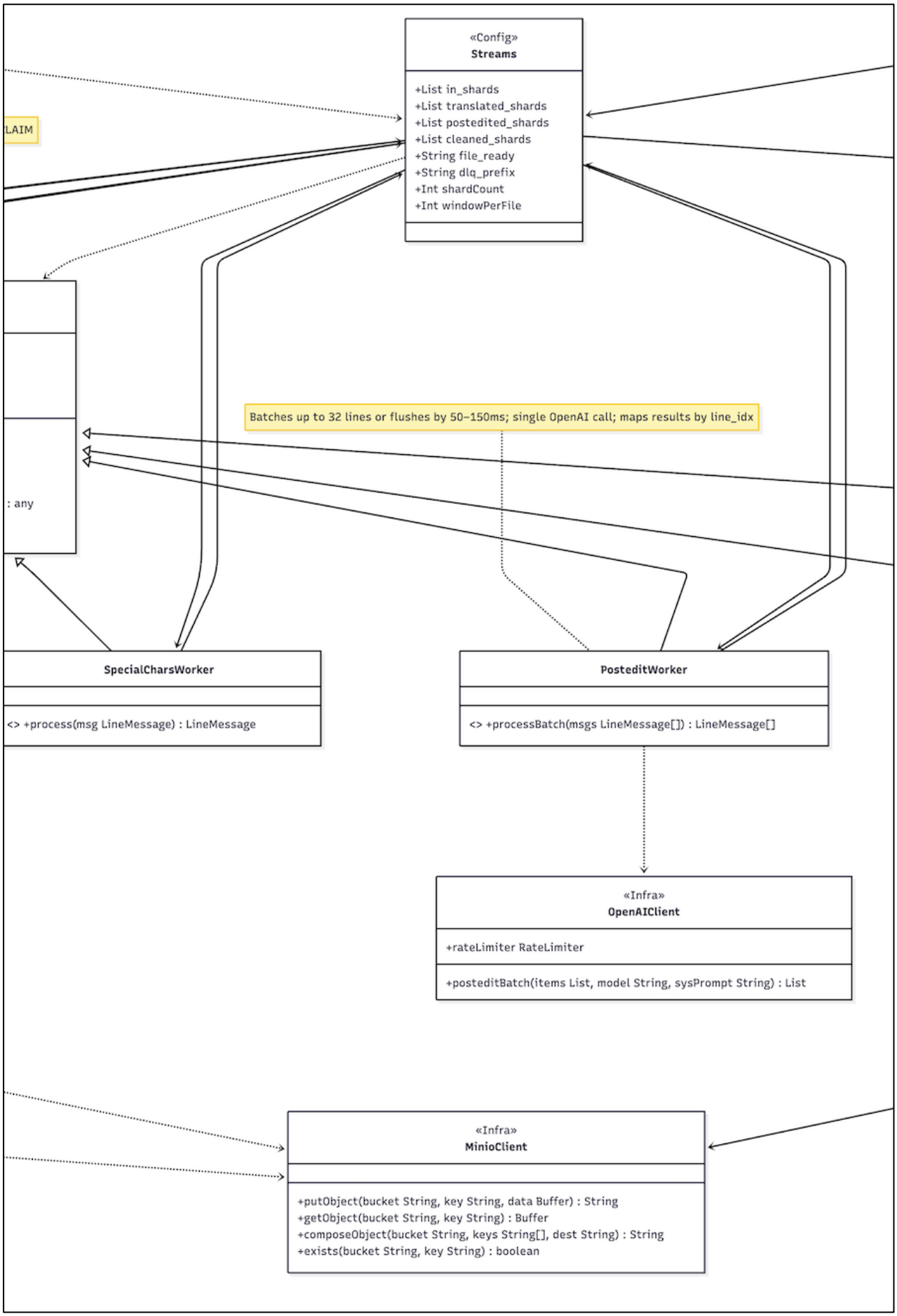


Рисунок 2.15 – Діаграма класів (6 частина)

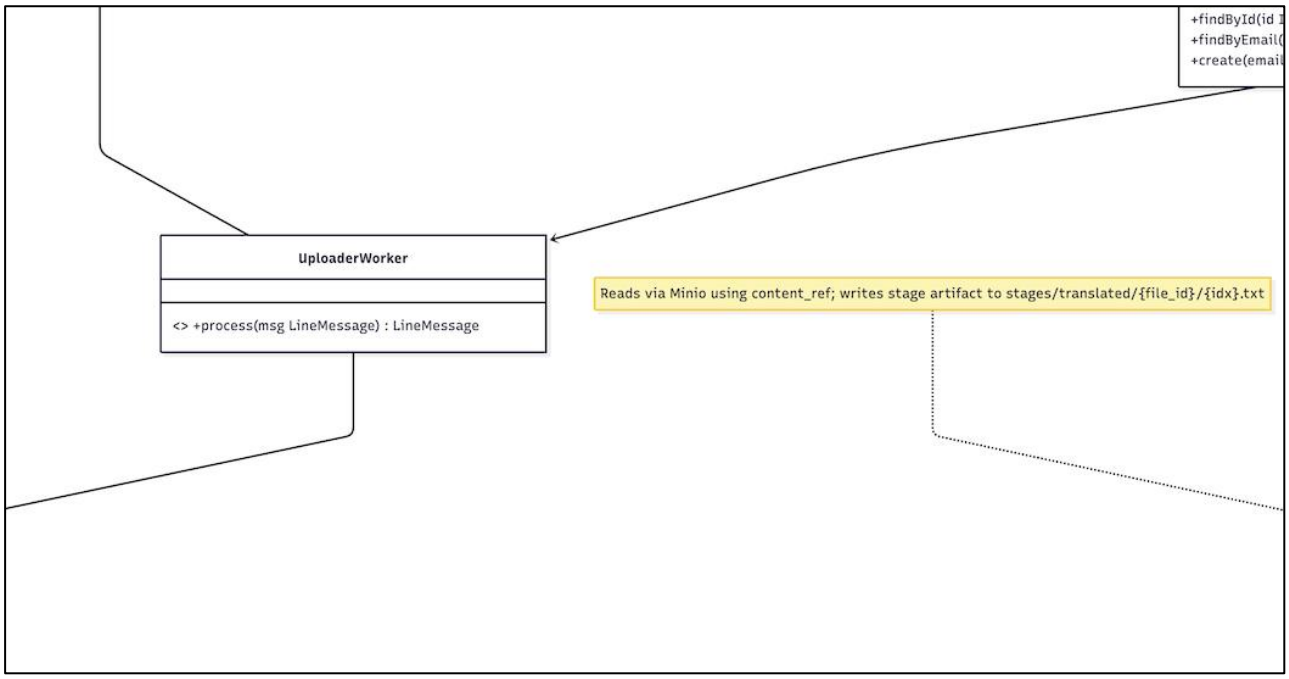


Рисунок 2.16 – Діаграма класів (7 частина)

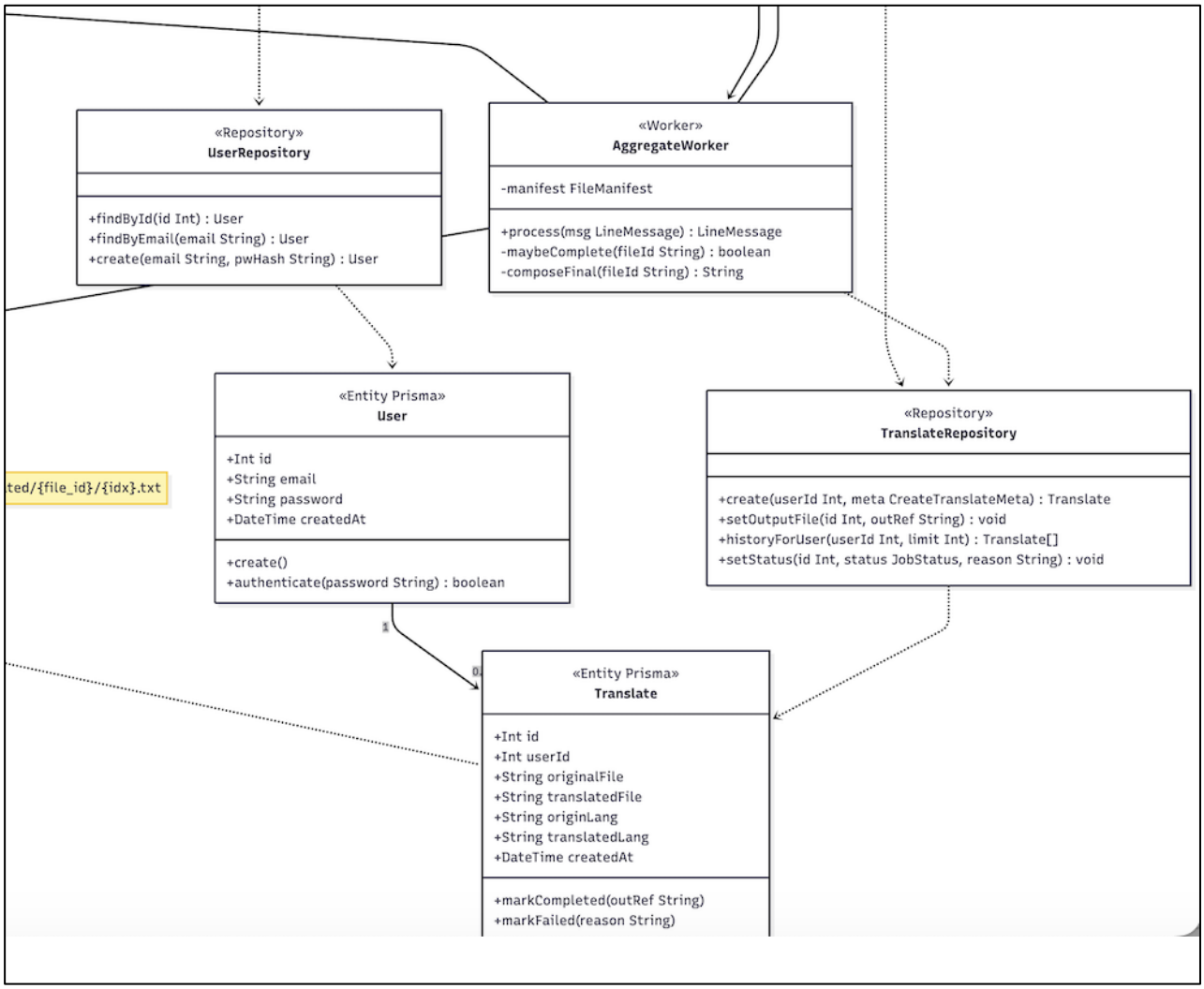


Рисунок 2.17 – Діаграма класів (8 частина)

2.8 Діаграма розгортання (Deployment Diagram)

Діаграма розгортання – це графік, що показує вузли, які були обчислені при запущеній програмі. Компоненти – це вигляд робочого екземпляра блоку коду. Діаграма створена для показу програмного забезпечення у створеному сервісі. За допомогою даної діаграми можна отримати повну картину того, як буде працювати програма на апаратному рівні [5, 6].

Відображення структури розгортання системи у середовищі Docker Compose наведено на рисунках 2.18 – 2.20:

– усі сервіси розгортаються в окремих контейнерах;
– redis, PostgreSQL і MinIO використовують постійні томи зберігання;
– взаємодія між контейнерами відбувається через спільну мережу app-network;
– браузер користувача підключається до web-app (порт 3000) та websocket-service (порт 3001), далі запити надсилаються до api-service [12].

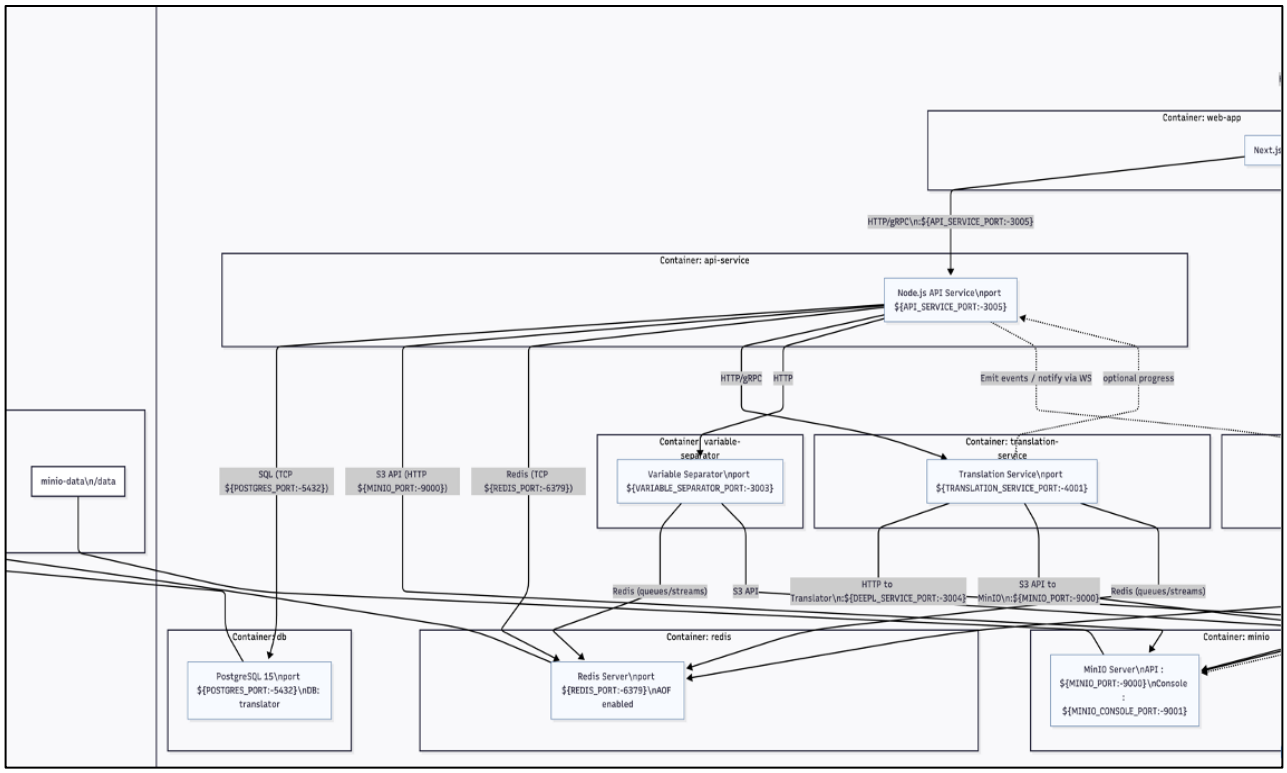


Рисунок 2.18 – Діаграма розгортання («центральна» частина)

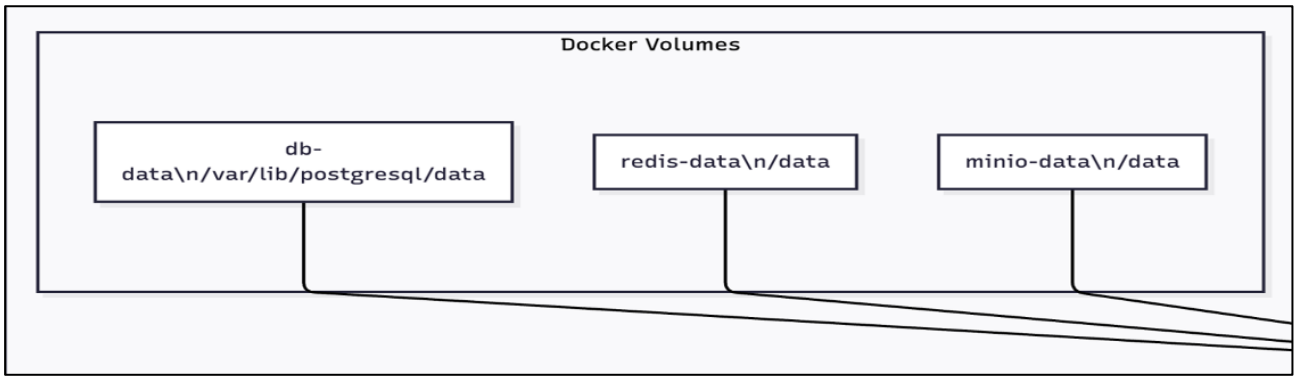


Рисунок 2.19 – Діаграма розгортання («ліва» частина)

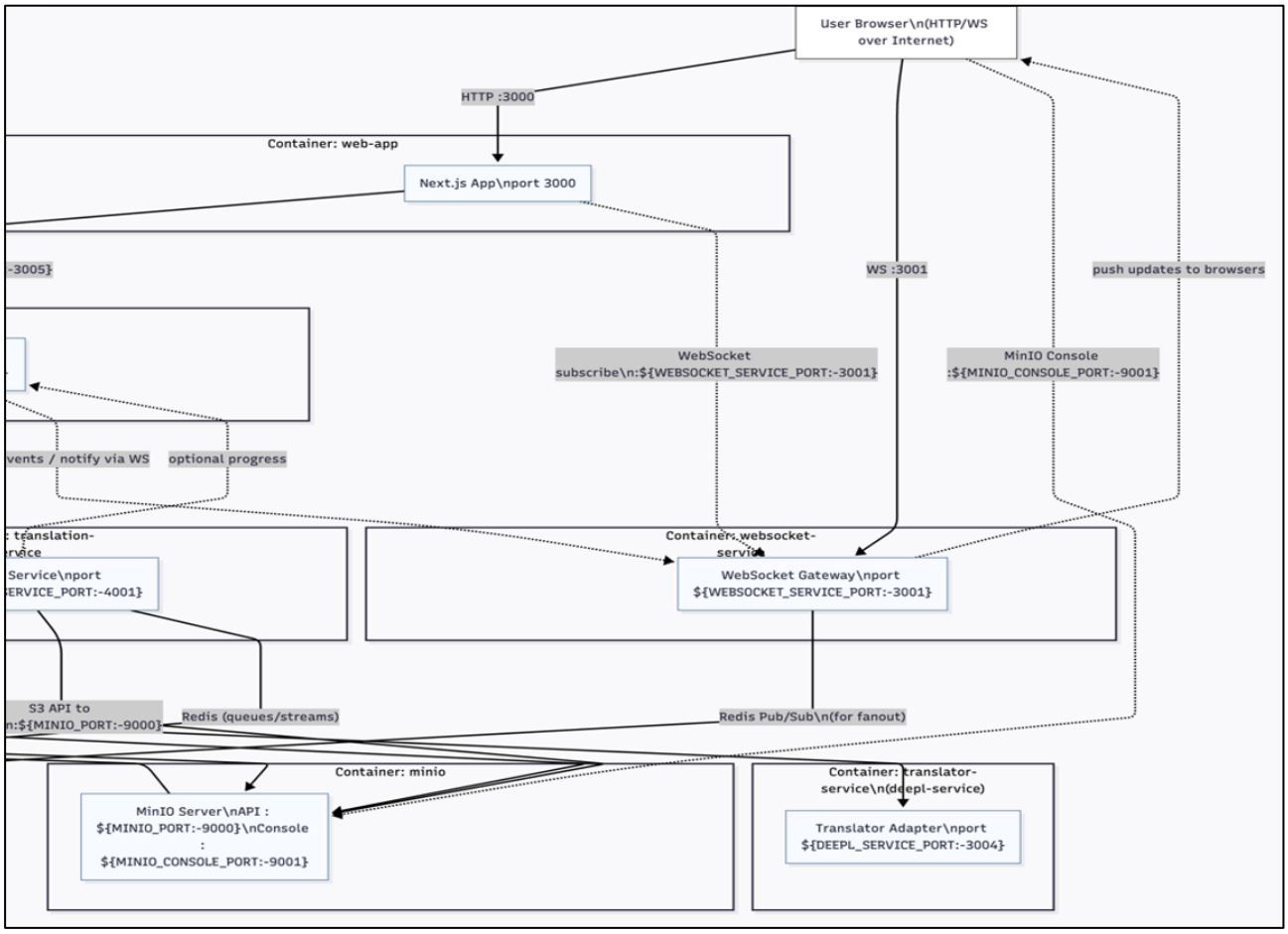


Рисунок 2.20 – Діаграма розгортання («права» частина)

2.9 Висновки до другого розділу

У цьому розділі було спроєктовано архітектуру вебсистеми для автоматизованого перекладу та постредакції локалізацій модів до ігор студії Paradox Interactive. Сформовано цілісну модель застосунку, засновану на

мікросервісному підході з використанням Redis Streams як основного каналу взаємодії між компонентами.

Було обґрунтовано вибір ключових технологій:

– Node.js та TypeScript – як сучасна платформа для побудови високопродуктивних серверних застосунків з коротким циклом розробки;
– Redis – як універсальний механізм кешування та обробки подій, що забезпечує асинхронність і стійкість системи;
– PostgreSQL – як стабільна реляційна база даних для зберігання користувачів, історії перекладів та метаданих.

Було розроблено структуру бази даних, визначено потоки даних між сервісами та побудовано UML-діаграми, які описують взаємодію компонентів, класи доменної моделі, процеси виконання та середовище розгортання.

Запропонована архітектура поєднує в собі асинхронну модель обробки, горизонтальне масштабування та толерантність до збоїв.

Завдяки Redis Streams забезпечується обробка великої кількості рядків у паралельному режимі, а контейнеризація (Docker) гарантує ізоляцію мікросервісів і простоту розгортання.

Таким чином, створена архітектура є гнучкою, масштабованою та придатною до промислового використання. Вона формує надійний фундамент для подальшої реалізації системи, описаної в наступному розділі.

3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Опис розробленого програмного продукту

Додаток Paradox Game Translation App – це інструмент, спеціально розроблений для перекладу ігор від компанії Paradox Interactive (таких як Hearts of Iron IV, Europa Universalis IV тощо). Метою додатку є забезпечення високоякісних перекладів із збереженням специфічних форматів та змінних, що використовуються в іграх. Завдяки поєднанню швидкості й точності API DeepL із контекстуальним розумінням OpenAI, додаток пропонує інноваційний підхід до локалізації контенту.

Основні можливості додатку:

– розумна обробка змінних: автоматичне збереження змінних і форматувальних кодів, характерних для ігор Paradox Interactive;
– гібридний переклад: базовий переклад здійснюється за допомогою DeepL, а додаткове постпроцесування можливе завдяки OpenAI;
– реальний час: оновлення статусу перекладу у реальному часі через WebSocket;
– підтримка YAML: правильна обробка файлів локалізації у форматі YAML, включаючи коментарі;
– мікросервісна архітектура: масштабована структура додатку з використанням Redis для управління чергами;
– сучасний інтерфейс користувача: додаток побудований на базі Next.js та TypeScript, що забезпечує інтуїтивний і зручний користувацький досвід.

Призначення та цільова аудиторія. Інструмент призначений для спільнот моддерів і локалізаторів ігор Paradox Interactive. Він допомагає значно скоротити час і підвищити якість перекладу, автоматизуючи рутинні завдання й мінімізуючи помилки, пов’язані із заміною змінних або спеціальних кодів. Загальні технічні характеристики (необхідні залежності): Node.js версії 18,

Docker Compose для запуску мікросервісів, API-ключі DeepL (обов’язково) та OpenAI (опційно). Сервіси додатку:

– Frontend – клієнтська частина з сучасним дизайном, розроблена на Next.js і TypeScript;
– WebSocket Service – забезпечує реальну комунікацію між клієнтом і сервером, управління чергами задач;
– Variable Separator – сервіс для вилучення, хешування та збереження змінних під час перекладу;
– DeepL Service – інтеграція з API DeepL швидкий переклад тексту;
– Translation Service – сервіс для опціонального постпроцесингу за допомогою OpenAI;
– Redis – система управління повідомленнями та чергами для зв’язку між сервісами.

Архітектурні особливості, що додаток має модульну структуру, яка складається з мікросервісів. Дозволяє масштабувати систему та легко додавати нові функції. Використання Redis як системи управління чергами забезпечує високу продуктивність і стабільність у роботі. Мови які підтримує додаток, доступні через DeepL API: англійську, німецьку, французьку, іспанську, польську та багато інших.

Ключові переваги:

– простота інтеграції в робочі процеси локалізації завдяки підтримці YAML-файлів;
– зручний інтерфейс для відстеження прогресу перекладу;
– збереження всіх специфічних форматувальних елементів, що гарантує коректну роботу гри після локалізації.

Додаток Paradox Game Translation App є ефективним рішенням для автоматизації перекладу ігрових модифікацій, яке забезпечує високу якість і точність результатів, зменшуючи витрати часу та людських ресурсів.

3.2 Реалізація основних модулів та функціоналу

Додаток Paradox Game Translation App має свій текстовий редактор, що дозволяє зручно редагувати текст. Редактор прекрасно побудований із застосуванням React і TypeScript, що забезпечує швидкість та масштабованість. Реалізований як компонент RowTextArea, що надає зручний інтерфейс. Використовується для редагування тексту з урахуванням специфіки змінних і форматування в іграх Paradox Interactive [16, 17].

Ключові властивості:

– простота у використанні, звичайне текстове поле з розширеним функціоналом;
– збереження змінних, змінні та спеціальні коди форматування (§Y, §G тощо) зберігаються автоматично;
– інтеграція перекладу, можливість виклику перекладацького сервісу безпосередньо з редактора;
– адаптивний розмір, висота текстового поля автоматично змінюється залежно від обсягу тексту;
– стилізація, додаткові стилі для відображення статусу перекладу.

Основний код реалізації компоненту RowTextArea наведено на рисунку 3.1 [18].

```
const handleChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {
  const newText = e.target.value;
  onTextChange(newText);
};
useEffect(() => {
  if (textareaRef.current) {
    textareaRef.current.style.height = 'auto';
    textareaRef.current.style.height = `${textareaRef.current.scrollHeight}px`;
  }
}, [text]);
return (
  <textarea
    ref={textareaRef}
    value={text}
    onChange={handleChange}
    className={`textarea ${isTranslated ? 'translated' : ''}`}
```

Рисунок 3.1 – Реалізація компоненту

Технічні переваги RowTextArea:

– продуктивність: легка реалізація без використання важких залежностей та швидкий рендеринг та оновлення;
– інтеграція – це пряме підключення до системи перекладу;
– користувацький досвід, інтуїтивний інтерфейс із візуальним зворотним зв’язком та адаптивний розмір текстового поля для зручності редагування.

Поточні обмеження: відсутність трекінгу вибору тексту. А саме в даний час редактор не зберігає інформацію про вибраний текст (див. рис. 3.2).

```
const [selection, setSelection] = useState<{
  start: number;
  end: number;
} | null>(null);
```

Рисунок 3.2 – Приклад поточних обмежень

Пропоновані покращення. Трекінг вибору тексту (див. рис. 3.3).

```
const handleSelect = (e: React.SyntheticEvent<HTMLTextAreaElement>) => {
  const target = e.target as HTMLTextAreaElement;
  setSelection({
    start: target.selectionStart,
    end: target.selectionEnd,
    text: text.substring(target.selectionStart, target.selectionEnd),
  });
};
```

Рисунок 3.3 – Вибір тексту

Підсвічування змінних, проходить одним рядком: const highlightVariables = (text: string) => { return text.replace(/\$[a-zA-Z0-9_]+\$/g, match => `<span class="variable">${match}</span>`);};. Текстовий редактор додатку Paradox Game Translation App є оптимальним рішенням для задач локалізації, забезпечуючи баланс між функціональністю, продуктивністю та зручністю.

Незважаючи на дрібні обмеження, поточна реалізація дозволяє ефективно обробляти тексти та інтегрувати їх із системою перекладу.

Парсер. Додаток Paradox Game Translation App використовує спеціалізований YAML-парсер для обробки файлів локалізації ігор. Цей парсер реалізований на основі TypeScript, забезпечуючи високу продуктивність, гнучкість і підтримку специфічного форматування.

Основні функції:

– ефективна обробка рядків: обробка файлів построково; пропуск порожніх рядків, коментарів та заголовків мов; мінімальне споживання пам’яті завдяки поетапній обробці;
– фільтрація рядків – пропускає непотрібні рядки за умовами;
– шаблонне розбиття – використання регулярних виразів для поділу рядків на коди та текст: const [code, text] = line.split(/:(.*)/s). Збереження форматування;
– менеджмент стану – інтеграція з контекстом файлів для управління мовами, рядками та файлом локалізації (див. рис. 3.4).

```
const {
    setFile,
    file,
    setOriginLang,
    setRows,
    setTargetLang
} = useContext(FileContext);
```

Рисунок 3.4 – Інтеграція управління

Переваги реалізації:

– продуктивність: асинхронне читання файлів і оптимізована обробка даних;
– масштабованість: чітка структура коду та розподіл обов’язків між компонентами;
– зручність використання: швидке завантаження файлів, вибір мов і чітке відображення статусу помилок;

– захист від помилок: фільтрація некоректних рядків: if (!code || !text) continue; // Пропустити некоректні рядки.

Інтеграція з перекладацькою системою:

– обробка типів файлів: приймає файли у форматі YAML: <FileInput accept="application/x-yaml" aria-label={en.fileInput.selectFile}/>;

– вибір мови: інтерактивний вибір мов перекладу (див. рис. 3.5).

```
<Select
  value={selectedLanguage}
  onValueChange={handleLanguageChange}
  disabled={!file}
>
  <SelectContent>
    {Object.values(LANGUAGE_CODES).map((lang) => (
      <SelectItem key={lang} value={lang}>
        {en.languageSelect[lang.toLowerCase()] || lang}
      </SelectItem>
    ))}
  </SelectContent>
</Select>
```

Рисунок 3.5 – Код системи перекладу

YAML-парсер додатку Paradox Game Translation App є високопродуктивним рішенням для обробки файлів локалізації. Незважаючи на вузьку спеціалізацію, він чудово інтегрується в систему перекладу та відповідає всім вимогам Paradox Interactive щодо збереження структури та форматування.

Архітектура мікросервісів додатку Paradox Game Translation App забезпечує ефективне розподілене оброблення даних, високу продуктивність і масштабованість. Одним із ключових елементів системи є WebSocket-сервіс, який дозволяє здійснювати обробку завдань у реальному часі та керувати чергами за допомогою Redis.

Основні компоненти архітектури WebSocket розглянемо в два кроки.

Крок 1: конфігурація сервісу. const VARIABLE_SEPARATOR_URL = process.env.VARIABLE_SEPARATOR_URL || 'http://localhost:3002';

const DEEPL_SERVICE_URL = process.env.DEEPL_SERVICE_URL || 'http://localhost:3004'; const REDIS_URL = process.env.REDIS_URL || 'redis://localhost:6379'; const redis = new Redis(REDIS_URL); const subscriber = new Redis(REDIS_URL); const publisher = new Redis(REDIS_URL).

Крок 2: система черг. const separatorQueue = new Queue('separator', REDIS_URL); const deeplQueue = new Queue('deepl', REDIS_URL); const restoreQueue = new Queue('restore', REDIS_URL).

Інтеграція Redis. Система Pub/Sub. WebSocket підписується на оновлення статусу перекладу через Redis. subscriber.subscribe('translation-status'); subscriber.on('message', (channel, message) => {if (channel === 'translation-status');

Обробка завдань із черг включає декілька етапів: розділення змінних, переклад, відновлення форматування (переваги мікросервісної архітектури).

Мікросервісна архітектура з використанням WebSocket і Redis у додатку Paradox Game Translation App забезпечує високу продуктивність, масштабованість і надійність. Вона дозволяє обробляти завдання паралельно, підтримувати реальний час і зменшувати затримки, що є критичним для ефективної локалізації ігрового контенту.

Сервіс розділення змінних – це критичний компонент додатку Paradox Game Translation App, розроблений для забезпечення цілісності та відповідності форматування тексту й змінних, характерних для ігор під час процесу перекладу. Нижче наведено детальний опис його функціоналу, інновацій та практичного застосування.

Основні компоненти – це константи та шаблони – VARIABLE_PATTERN: виявляє змінні у тексті у форматі [VariableName]. SPECIAL_CODES: обробляє специфічні коди форматування, такі як §R, §G, що визначають стилі та кольори тексту. Ці коди сортуються за довжиною для оптимальної обробки (див. рис. 3.6).

Генерація хешів (див. рис. 3.7). Криптографічний MD5-хеш у поєднанні з UUID забезпечує унікальні ідентифікатори для змінних та кодів, знижуючи ризик колізій.

```
    const VARIABLE_PATTERN = /\[([^\]]+)\]/g;
    const SPECIAL_CODES = [
      "§!", "§C", "§L", "§W", "§B", "§G",
      "§R", "§b", "§g", "§Y", "§H", "§T"
].sort((a, b) => b.length - a.length);
```

Рисунок 3.6 – Код сортування

```
    private generateHash(text: string): string {
      return createHash('md5')
        .update(text + uuidv4())
        .digest('hex')
        .slice(0, 8);
  }
```

Рисунок 3.7 – Код генерації хешів

Основний функціонал. Виділення змінних це коли витягує змінні та коди форматування з тексту, замінюючи їх унікальними хешами. Відновлення змінних коли проходить відновлення змінної та коду форматування до їх початкового вигляду для точного відображення. По міркам інженерного рішення можна поділити на наступний список:

– розумна обробка шаблонів: попереднє сортування та екранування шаблонів регулярних виразів покращують ефективність та уникають конфліктів. SPECIAL_CODES.sort((a, b) => b.length - a.length);
– унікальне хешування: поєднання MD5-хешування та UUID гарантує, що ідентифікатори є одночасно короткими та унікальними;
– типобезпечність: забезпечує надійну розробку завдяки чітким інтерфейсам.

Ключові можливості:

– підтримка формату Paradox: зберігає коди форматування з префіксом § та змінні, підтримуючи їхню цілісність;
– оптимізація продуктивності: попередня компіляція регулярних виразів і обробка тексту за один прохід мінімізують споживання пам’яті та підвищують швидкість;

– двостороння обробка забезпечує безперешкодне витягування та відновлення змінних і кодів форматування.

Можна переглянути приклади використання в наступному переліку:

– збереження кольорових кодів: вхідні дані: "§RЧервоний текст§!". Оброблений: "{hash1}Червоний текст{hash2}". Відновлений: "§RЧервоний текст§!";
– обробка змінних: вхідні дані: "Привіт, [Player.Name]!". Оброблений: "Привіт, {hash1}!". Відновлений: "Привіт, [Player.Name]!".

Сервіс розділення змінних є основою перекладацького додатку, забезпечуючи точну та ефективну обробку текстових елементів, характерних для ігор Paradox Interactive, із збереженням їх унікальних форматів. Його переваги це надійність, продуктивність та легкість підтримки.

Сервіс DeepL у рамках додатку Paradox Game Translation App забезпечує високу якість перекладу текстів з використанням штучного інтелекту. Його структура побудована таким чином, щоб підтримувати гнучкість, масштабованість і збереження специфічних форматувань, необхідних для ігрових даних. Відображення мовних кодів Для забезпечення правильного перекладу текстів використовується карта мовних кодів, яка відображає формат, підтримуваний іграми Paradox, у відповідний формат API DeepL. Наприклад, код l_english відповідає EN-GB, а l_simplified_chinese – ZH-HANS. Це дозволяє ефективно працювати з локалізаціями різних мов. Типи даних і запитів Сервіс працює із суворо визначеними структурами запитів і відповідей, забезпечуючи типобезпечність. Запит включає текст і цільову мову, а у відповідь повертається перекладений текст або повідомлення про помилку.

Основний функціонал. Конвертація мовних кодів, як основна функція сервісу (конвертація кодів локалізації ігор у формат, зрозумілий DeepL API). Якщо код мови не підтримується, сервіс повідомляє про помилку. Процес перекладу Сервіс виконує переклад текстів у кілька етапів. Приймає запит із текстом і мовним кодом. Перевіряє валідність введених даних. Надсилає запит до DeepL API, використовуючи відповідний ключ доступу. Повертає перекладений текст клієнту.

Збереження форматування Однією з ключових функцій є збереження форматування тексту, включаючи кольорові коди та спеціальні символи. Це критично для локалізації ігрових даних, де будь-яке пошкодження форматування може призвести до помилок у грі.

Готові інженерні рішення:

– валідація введення: сервіс перевіряє, чи всі необхідні дані (текст, мовний код) були передані (у разі відсутності таких даних система повертає користувачу детальне повідомлення про помилку);
– обробка помилок: якщо під час обробки запиту виникає помилка (наприклад, недійсний ключ API або проблеми з підключенням), сервіс гарантує обробку помилки та надає відповідний відгук клієнту (логування дозволяє швидко ідентифікувати проблеми);
– масштабованість: завдяки використанню зовнішнього API сервіс може обробляти значні обсяги запитів паралельно, забезпечуючи швидкість і стабільність роботи.

Ключові можливості розробки це розширена підтримка мов. Понад 30 мовних пар із підтримкою регіональних варіантів. Збереження специфічного форматування, підтримка кодів кольорів та інших стилів. Захист від помилок і валідність даних. Просте додавання нових мовних кодів і налаштування API. (гнучкість інтеграції).

Можна розібрати декілька прикладів використання. Перший простий переклад, наш користувач може надіслати наступний текст, наприклад: “Hello world”, із вказаною мовою перекладу. Отримати результат: “Bonjour le monde”, що притаманно французької мови. Другий приклад збереження стилю тексту зі спеціальними кодами, наприклад: “§RWarning: Critical Error§!”. Наш переклад збереже форматування, перетворюючи його у відповідний стиль іншою мовою.

Переглянуті приклади призводять на домисли наступних переваг:

– захищеність: використання змінних середовища для ключів API забезпечує безпеку даних;
– гнучкість: легка адаптація під нові вимоги чи локалізації;

– ефективність: швидка обробка запитів завдяки оптимізованій взаємодії з API.

Реалізація DeepL Service в додатку Paradox Game Translation App дозволяє забезпечити ефективну локалізацію тексту з урахуванням особливостей ігрових даних, що робить її важливим інструментом для спільноти моддерів.

3.3 Налаштування Docker контейнерів для розгортання

Призначення для уніфікованого розгортання системи перекладів модів Paradox застосовано контейнеризацію на базі Docker та оркестрацію за допомогою Docker Compose. Архітектура складається з окремих сервісів (бази даних, черги, об’єктного сховища, бекенд-API, сервісів перекладу й пост-обробки, WebSocket-шлюзу та вебклієнта), об’єднаних у спільну мережу app-network. Такий підхід забезпечує ізоляцію середовищ і відтворюваність збірок, явне декларування залежностей (портів). Відокремлені політики перезапуску та оновлення. Масштабування на рівні сервісів. Фрагменти конфігурації сокетів (infra-as-code) (див. рис. 3.8).

```
services:
  db:              # PostgreSQL 15 (alpine)
  redis:           # Redis для керування чергою
  minio:           # S3-сумісне сховище артефактів
  api-service:     # NestJS API (ядро доменної логіки)
  translation-service:   # сервіс пост-обробки перекладів
  web-app:         # Next.js веб-застосунок
  translator-service:# Microsoft/DeepL адаптер
  websocket-service: # шлюз подій/статусів через WebSocket
  variable-separator:# препроцесор спеціальних змінних (профіль 'full')

networks:
  app-network:
```

Рисунок 3.8 – Код конфірурації (infra-as-code)

Усі сервіси підключені до місткової мережі app-network. Взаємодія між контейнерами відбувається за DNS-іменами сервісів (наприклад, db:5432,

minio:9000). Зовнішня експозиція обмежена портами, явно зазначеними у секції ports. Така схема мінімізує поверхню атаки та полегшує розмежування доступів.

Сховище даних та персистентність, діляться:

– PostgreSQL: db-data для журналів і файлів таблиць;

– Redis: redis-data для snapshot-ів (RDB/AOF за потреби);

– MinIO: minio-data для об’єктів (великі файли локалізацій/архівів).

Завдяки іменованим томам дані зберігаються поза життєвим циклом контейнерів.

Керування конфігураціями проходить з підключення секції env_file в окремі .env для кожного сервісу (мінімізація повторення та випадкового витоку). Рекомендують не зберігати .env у VCS використовувати dotenv-сховище або менеджери секретів. Для продакшну прибрати публічні ports там, де доступ потрібен лише зсередини мережі (див. рис. 3.9). Саме для Redis/MinIO встановити автентифікацію (пароль Redis, MINIO_ROOT_USER/SECRET тощо).

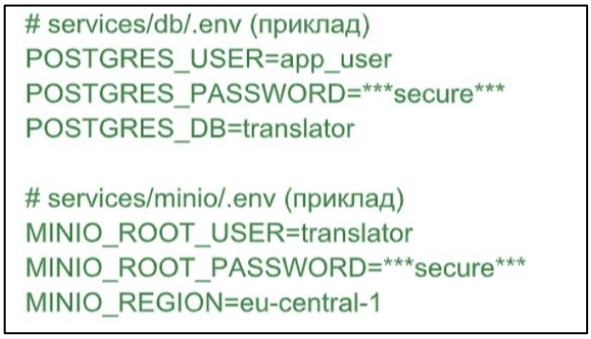


Рисунок 3.9 – Код приклад .env для db і minio

Порядок запуску depends_on забезпечує послідовність старту контейнерів. Для підвищення надійності у продакшні доцільно додати healthcheck (наприклад, для db, minio, api-service) і використовувати depends_on: condition: service_healthy.

Процедура розгортання починається з підготовки оточення.

Крок 1: встановити Docker Engine та Docker Compose сумісних версій.

Крок 2: створити .env файли для кожного сервісу з необхідними секретами та параметрами.

Крок 3: ініціалізувати каталоги для іменованих томів. Запуск інфраструктури починається зі збирання образів (якщо є локальні Dockerfile) «docker compose build» та старту у фоновому режимі «docker compose up -d». Пройти перевірку стану можна за наступним алгоритмом: docker compose ps, docker compose logs -f api-service, docker compose logs -f web-app. І головне перевірити доступність:

– вебклієнта: http://localhost:3000;
– PI: http://localhost:3005;
– MinIO Console: http://localhost:9001.

Також є міграції БД, якщо бекенд використовує ORM (наприклад, Prisma), міграції виконуються всередині api-service: docker compose exec api-service npx prisma migrate deploy. Початкові дані (seeds) можуть бути застосовані через окремий скрипт у контейнері.

Оновлення сервісів виконується на рівні окремих образів: docker compose pull api-service та docker compose up-d api-service. У CI/CD (GitHub Actions/GitLab CI) доцільно передбачити етапи лінтингу, збірки Docker-образів, прогону інтеграційних тестів у тимчасовому compose-стенді та поштовху образів у реєстр.

3.4 Тестування функціоналу системи

Додаток вирізняється своєю простотою та мінімалістичним дизайном, що робить його зручним у використанні. Основний інтерфейс включає одну сторінку з декількома кнопками, що забезпечують інтуїтивний процес локалізації. На головній сторінці користувачеві пропонується завантажити файл локалізації та обрати цільову мову перекладу. Візуальний приклад інтерфейсу наведено нижче (див. рис. 3.10).

Після завантаження файлу локалізації він автоматично проходить обробку (парсинг), після чого користувач отримує доступ до окремих рядків для

редагування. Кожен рядок тексту представлений разом із його унікальним ідентифікатором, що спрощує навігацію та управління даними.

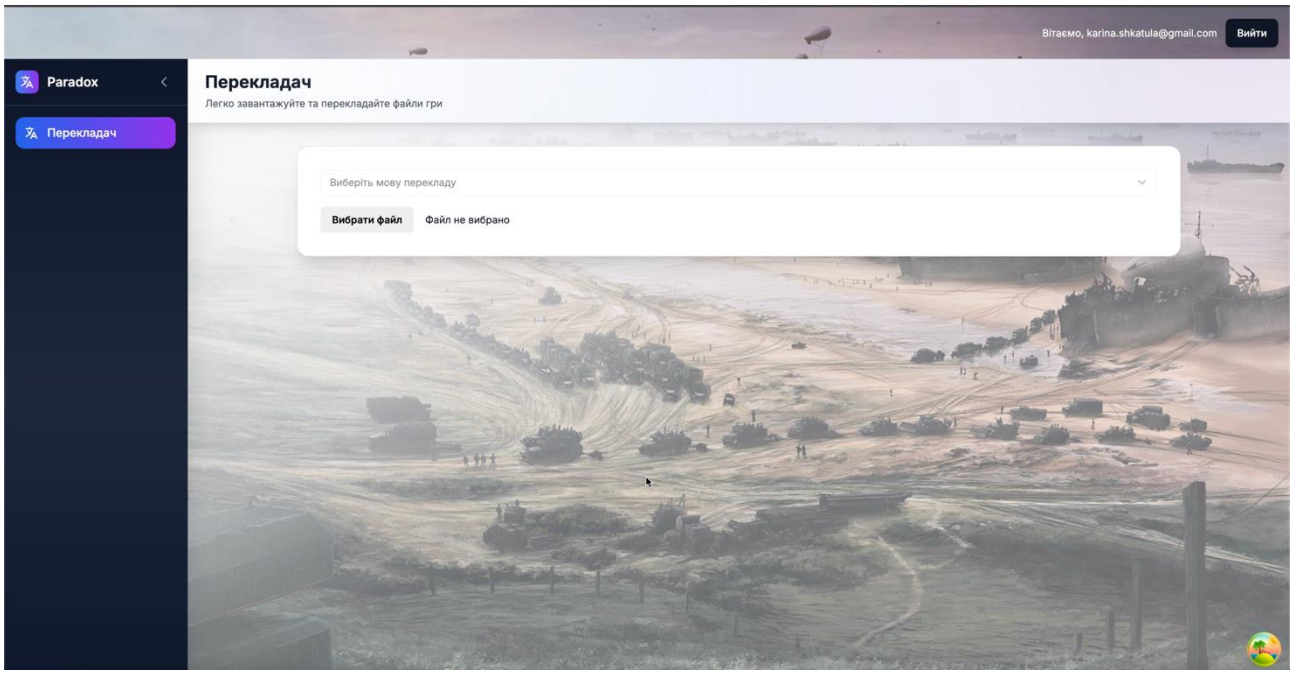


Рисунок 3.10 – Головний інтерфейс до вибору файлу

Інтерфейс також надає кнопки перекладу, які дозволяють надсилати текст на сервер для обробки. Можливі два режими: переклад окремого рядка або всіх рядків одночасно. Переклад виконується через сокет-з’єднання, що забезпечує швидку взаємодію між клієнтом та сервером. Після завершення обробки перекладені результати автоматично відображаються у правій колонці редактора. Нижче представлено приклад сторінки після завантаження файлу для редагування, а також вигляд інтерфейсу після завершення перекладу (див. рис. 3.11).

Після завершення перекладу користувач має можливість експортувати файл, натиснувши відповідну кнопку експорту. Система автоматично генерує файл з валідною назвою та у правильному форматі, що гарантує його повну сумісність із модами Paradox Interactive. Ця функція значно спрощує інтеграцію перекладених даних до ігрового середовища.

Файл включає всі оброблені рядки з перекладом, збереженням змінних та форматувальних кодів.

Нижче наведено приклад успішно перекладеного файлу, зокрема на португальську мову, із збереженням кольорових кодів та структури тексту.

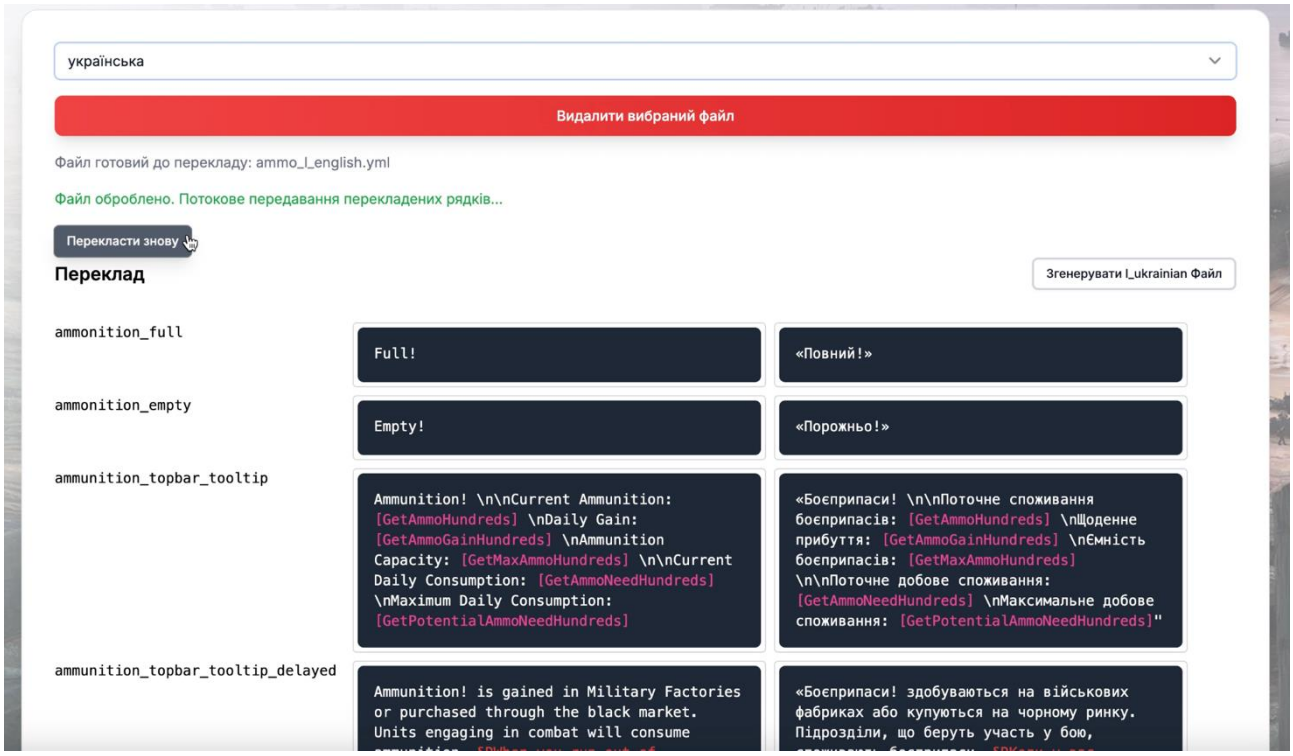


Рисунок 3.11 – Головна сторінка після завершення перекладу

3.5 Висновки до третього розділу

У процесі виконання роботи було досягнуто всіх поставлених практичних цілей. Здійснено детальний аналіз потреб користувачів та особливостей ігрової спільноти Paradox Interactive, що підтвердило необхідність створення нового інструменту для перекладу модифікацій. Аналіз існуючих рішень виявив суттєві недоліки, які могли бути подолані шляхом розробки вебдодатку.

Розробка робочого додатку включала вибір відповідного функціоналу та архітектури. На основі проведених досліджень було вирішено використовувати сучасні технології, такі як Node.js, Redis, DeepL API та OpenAI API, що забезпечують ефективність, гнучкість та масштабованість системи. Основною перевагою розробленого вебдодатку є його доступність через браузер, що усуває обмеження платформ та складності встановлення.

Додаток реалізовано з використанням мікросервісної архітектури, що дозволяє обробляти великі обсяги даних, зберігаючи стабільність навіть за високого навантаження. Інтеграція сокет-з'єднань для передачі даних у реальному часі забезпечує швидкий та зручний процес перекладу. Використання системи черг Redis дозволяє обробляти запити паралельно, що значно підвищує продуктивність.

Розроблено механізми для збереження специфічного форматування текстів та правильного оброблення змінних, що є критично важливим для ігор Paradox Interactive. Збереження кольорових кодів та інших стилів забезпечує коректну роботу модифікацій після інтеграції.

Створений додаток Paradox Game Translation App відповідає всім технічним вимогам, які були поставлені на початку проєкту. Він є ефективним інструментом для спільноти моддерів, що дозволяє значно спростити процес перекладу модифікацій, розширити доступність цих інструментів та підтримати активність гравців. Результати роботи свідчать про практичну цінність проєкту для підтримки розвитку ігрової індустрії.

Також було проведено детальне дослідження інтеграції мікросервісної архітектури та сокетів, які були доведені до максимальної функціональності в рамках обраної архітектури. Було виявлено, що мікросервіси та сокети рідко використовуються разом через складність реалізації та необхідність значних ресурсів для підтримки такого підходу. Незважаючи на це, їх поєднання має певні переваги, такі як висока швидкість обміну даними у реальному часі та можливість масштабування окремих компонентів системи.

Однак, цей підхід вирізняється високим рівнем складності розробки та потенційним оверінжирингом, що робить його недоцільним у більшості типових задач. У випадках, коли потрібна максимально ефективна платформа, зазвичай використовуються технології, які краще відповідають вимогам, такі як ASP.NET або Go. Теоретично та практично було доведено, що подібний підхід може бути реалізований на Node.js, проте в більшості реальних випадків застосування цього підходу не є оптимальним через його складність і витрати на підтримку.

ВИСНОВКИ

Вході роботи магістра було досягнуто основну мету: реалізовано сервіс перекладу з використанням мікросервісної архітектури для перекладу модифікацій ігор Paradox Interactive.

Переклад модів на різні мови вручну є тривалим та складним процесом, особливо коли мова йде про великі модифікації з численними текстовими блоками. З огляду на це, існує потреба в автоматизації процесу перекладу з використанням сучасних технологій обробки природної мови. В даному дослідженні розглядається розробка вебдодатку-помічника перекладача, який використовує DeepL API та OpenAI API для локалізації модифікацій з будь-якої мови на будь-яку іншу з тих, що підтримуються в іграх компанії Paradox Interactive.

У першому розділі було проаналізовано характеристику предметної області, визначені вимоги до вебсервісу, огляд аналізу готових рішень та використаних технологій.

У другому розділі було розглянуто технічне завдання та вивчено архітектуру розробленого сервісу. Було розроблено структуру бази даних, визначено потоки даних між сервісами та побудовано UML-діаграми, які описують взаємодію компонентів, класи доменної моделі, процеси виконання та середовище розгортання. Перевірка працюючих платформ та сервісів завдяки яким створено технічну частину програмного продукту. Показано взаємодію певних компонентів та потоки даних інформаційної системи, різні діаграми, які описують розробку проєкту.

У третьому розділі було описано тестування вже готового сервісу, створений додаток Paradox Game Translation App відповідає всім технічним вимогам, які були поставлені на початку магістерської роботи. Він є ефективним інструментом для спільноти моддерів, що дозволяє значно спростити процес перекладу модифікацій, розширити доступність цих інструментів та підтримати

активність гравців. Результати роботи свідчать про практичну цінність проєкту для підтримки розвитку ігрової індустрії.

Також реалізований сервіс можна ще більше поліпшити в деяких місцях, а саме:

– підтримка ігор інших студій;
– підтримка перекладу деяких файлів одночасно;
– інтеграція з Steam API для швидкого заливання модів до майстерні;
– переписати весь сервіс на більш продуктивну платформу (ASP.NET, Spring Boot, тощо).

Даний проєкт можна розвивати і на далі, вийде декілька цікавих комерційних проєктів з витікаючими програмними продуктами. Звичайно врахувати економічні збитки та географічну територіальність.

Для повного економічного обґрунтування розробки сервісу перекладу модів Paradox необхідно включити всі елементи і головне, це врахування регіону. При розробці цього програмного продукту відразу треба було враховувати фактор воєнного часу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Документація Node.js. URL: https://nodejs.org/api (дата звернення: 18.03.2025).

2. Рекомендації з перекладу інтерфейсів Microsoft. URL: https://learn.microsoft.com/globalization/ (дата звернення: 20.03.2025).

3. Шаблони проєктування в TypeScript. URL: https://refactoring.guru/uk/design-patterns/typescript (дата звернення: 18.03.2025).

4. Basarat A. TypeScript Deep Dive. URL: https://basarat.gitbook.io/typescript/ (дата звернення: 20.03.2025).

5. Docker Compose документація. URL: https://docs.docker.com/compose/ (дата звернення: 21.04.2025).

6. Docker Hub – офіційні образи. URL: https://hub.docker.com/ (дата звернення: 20.03.2025).

7. Docker документація. URL: https://docs.docker.com/ (дата звернення: 30.05.2025).

8. Eric Elliott. Programming JavaScript Applications. O’Reilly Media, 2014. 254 p.

9. Fowler M. Microservices: a definition of this new architectural term. URL: https://martinfowler.com/articles/microservices.html (дата звернення: 30.10.2025).

10. i18next документація. URL: https://www.i18next.com/ (дата звернення: 21.09.2025).

11. JavaScript MDN Довідник. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 21.09.2025).

12. Kubernetes документація. URL: https://kubernetes.io/docs/ (дата звернення: 20.08.2025).

13. Simpson K. You Don’t Know JS. O’Reilly Media, 2015 – 2020.

14. Microservices.io – каталоги патернів мікросервісної архітектури. URL: https://microservices.io/ (дата звернення: 20.10.2025).

15. NestJS – Patterns & Techniques. URL: https://docs.nestjs.com/recipes (дата звернення: 20.10.2025).

16. NestJS CQRS Guide. URL: https://docs.nestjs.com/recipes/cqrs (дата звернення: 20.10.2025).

17. Next.js Internationalization Guide. URL: https://nextjs.org/docs/pages/building-your-application/routing/internationalization (дата звернення: 21.05.2025).

18. Next.js документація. URL: https://nextjs.org/docs (дата звернення: 25.05.2025).

19. Newman S. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2015. 280 p.

20. Node.js Best Practices. *GitHub*. URL: https://github.com/goldbergyoni/nodebestpractices (дата звернення: 24.05.2025).

21. OpenAPI (Swagger) документація. URL: https://swagger.io/specification/ (дата звернення: 20.10.2025).

22. React документація. URL: https://react.dev/learn (дата звернення: 20.10.2025).

23. Richards M. Microservices vs Service-Oriented Architecture. O'Reilly Media, 2020. 156 p.

24. Newman S. Monolith to Microservices. O'Reilly Media, 2019. 272 p.

25. Translators Without Borders – гайдлайн з перекладів. URL: https://translatorswithoutborders.org/ (дата звернення: 20.10.2025).

26. TypeScript документація. URL: https://www.typescriptlang.org/docs/ (дата звернення: 29.10.2025).

27. Weblate документація. URL: https://docs.weblate.org/en/latest/ (дата звернення: 20.09.2025).

ДОДАТОК А

Основний докер файл

```
x-worker-env: &worker-env
  REDIS_HOST: redis
  REDIS_PORT: 6379
  REDIS_SHARDS: ${REDIS_SHARDS:-64}
  STREAM_READ_COUNT: ${STREAM_READ_COUNT:-32}
  STREAM_BLOCK_MS: ${STREAM_BLOCK_MS:-1000}
  MAX_ATTEMPTS: ${MAX_ATTEMPTS:-5}
  WINDOW_PER_FILE: ${WINDOW_PER_FILE:-25}
  MINIO_ENDPOINT: http://minio:9000
  MINIO_ACCESS_KEY: ${MINIO_ACCESS_KEY:-minioadmin}
  MINIO_SECRET_KEY: ${MINIO_SECRET_KEY:-minioadmin}
  MINIO_BUCKET: ${MINIO_BUCKET:-translator}
  MINIO_REGION: ${MINIO_REGION:-us-east-1}
  INPUT_PREFIX: ${INPUT_PREFIX:-inline}
  OUTPUT_PREFIX: ${OUTPUT_PREFIX:-out}
  EXPORT_PREFIX: ${EXPORT_PREFIX:-exports}
  MINIO_SHARED_DIR: /exports
  MINIO_QUEUE_NAME: ${MINIO_QUEUE_NAME:-minio:jobs}
  MINIO_WORKERS: ${MINIO_WORKERS:-4}
  POSTEDIT_BATCH_SIZE: ${POSTEDIT_BATCH_SIZE:-32}
  POSTEDIT_FLUSH_MS: ${POSTEDIT_FLUSH_MS:-250}
  OPENAI_API_KEY: ${OPENAI_API_KEY:-demo}
  OPENAI_MODEL: ${OPENAI_MODEL:-gpt-4o-mini}
services:
  db:
    image: postgres:15-alpine
    restart: always
    environment:
      POSTGRES_USER: ${POSTGRES_USER:-user}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD:-password}
      POSTGRES_DB: ${POSTGRES_DB:-translator}
    volumes:
      - db-data:/var/lib/postgresql/data
    ports:
      - "${POSTGRES_PORT:-5432}:5432"
    networks:
      - app-network
```

```
redis:
  image: redis:alpine
  command: ["redis-server", "--appendonly", "yes", "--appendfsync", "everysec"]
  restart: always
  ports:
    - "${REDIS_PORT:-6379}:6379"
  volumes:
    - redis-data:/data
  networks:
    - app-network

minio:
  image: minio/minio:RELEASE.2024-08-03T04-33-23Z
  command: server /data --console-address ":9001"
  env_file:
    - ./services/minio/.env
  environment:
    MINIO_SERVER_URL: http://minio:9000
  ports:
    - "${MINIO_PORT:-9000}:9000"
    - "${MINIO_CONSOLE_PORT:-9001}:9001"
  volumes:
    - minio-data:/data
  networks:
    - app-network
  restart: unless-stopped

translate-service:
  build:
    context: .
    dockerfile: services/translate-service/Dockerfile
    args:
      NODE_ENV: development
  env_file:
    - ./services/translate-service/.env
  command: npm run start:dev
  environment:
    <<: *worker-env
    NODE_ENV: development
  depends_on:
    - redis
    - minio
```

```
    volumes:
      - ./services/translate-service:/app
      - ./packages:/packages
      - /app/node_modules
    networks:
      - app-network
  postedit-service:
    build:
      context: .
      dockerfile: services/postedit-service/Dockerfile
      args:
        NODE_ENV: development
    command: npm run start:dev
    environment:
      <<: *worker-env
      NODE_ENV: development
    depends_on:
      - redis
      - minio
    volumes:
      - ./services/postedit-service:/app
      - ./packages:/packages
      - /app/node_modules
    networks:
      - app-network
  variable-service:
    build:
      context: .
      dockerfile: services/variable-service/Dockerfile
      args:
        NODE_ENV: development
    command: npm run start:dev
    environment:
      <<: *worker-env
      NODE_ENV: development
    depends_on:
      - redis
      - minio
    volumes:
      - ./services/variable-service:/app
      - ./packages:/packages
      - /app/node_modules
```

```
  networks:
    - app-network
ingest-splitter:
  build:
    context: .
    dockerfile: services/ingest-splitter/Dockerfile
    args:
      NODE_ENV: development
  command: npm run start:dev
  environment:
    <<: *worker-env
    NODE_ENV: development
    SPLIT_MAX_LINES: ${SPLIT_MAX_LINES:-2000}
    SPLIT_MAX_BYTES: ${SPLIT_MAX_BYTES:-5242880}
    SPLIT_CONCURRENCY: ${SPLIT_CONCURRENCY:-2}
  depends_on:
    - redis
    - minio
    - db
  volumes:
    - ./services/ingest-splitter:/app
    - ./packages:/packages
    - ./exports:/exports
    - /app/node_modules
  networks:
    - app-network
minio-service:
  build:
    context: .
    dockerfile: services/minio-service/Dockerfile
    args:
      NODE_ENV: development
  command: npm run start:dev
  environment:
    <<: *worker-env
    NODE_ENV: development
    EXPORT_PREFIX: ${EXPORT_PREFIX:-exports}
  depends_on:
    - redis
    - minio
  volumes:
    - ./services/minio-service:/app
```

```
      - ./packages:/packages
      - ./exports:/exports
      - /app/node_modules
    networks:
      - app-network
  api-service:
    build:
      context: .
      dockerfile: dockerfiles/api-service.Dockerfile
    env_file:
      - ./services/api-service/.env
    command: npm run start:dev
    environment:
      <<: *worker-env
      NODE_ENV: development
    depends_on:
      - db
      - redis
      - minio
    ports:
      - "${API_SERVICE_PORT:-3005}:3005"
    volumes:
      - ./services/api-service:/app
      - ./packages:/packages
      - ./exports:/exports
      - /app/node_modules
    networks:
      - app-network
networks:
  app-network:
    driver: bridge
volumes:
  db-data:
  redis-data:
  minio-data:
```

ДОДАТОК Б

Лістинг коду файлу для перекладу

```
l_english:
 ammunition_value: "[GetAmmoFilling]"
 ammonition_full: "§GFull§!"
 ammonition_filling: "[?ammo_days_to_full|0G] §Gd§!"
 ammonition_dwindles: "[?ammo_days_to_empty|0R] §Rd§!"
 ammonition_empty: "§REmpty§!"
 ammunition_topbar_tooltip: "§HAmmunition§! \n\nCurrent Ammunition: [GetAmmoHundreds] \nDaily Gain: [GetAmmoGainHundreds] \nAmmunition Capacity: [GetMaxAmmoHundreds] \n\nCurrent Daily Consumption: [GetAmmoNeedHundreds] \nMaximum Daily Consumption: [GetPotentialAmmoNeedHundreds]"
 ammunition_topbar_tooltip_delayed: "§HAmmunition§! is gained in Military Factories or purchased through the black market. Units engaging in combat will consume ammunition. §RWhen you run out of ammunition your army receives attack and defese penalties up to a point where they are unable to fight!§!"
 ammonition_hundreds: "[?ammo_h|H]"
 ammonition_thousands: "[?ammunition_variable|H1]§HK§!"
 ammonition_max_hundreds: "[?ammunition_max_h|H]"
 ammonition_max_thousands: "[?ammunition_max|H1]§HK§!"
 ammonition_gain_hundreds: "[?ammo_daily_h|H0]"
 ammonition_gain_thousands: "[?ammunition_daily|H1]§HK§!"
 ammonition_need_hundreds: "[?ammo_need_h|H0]"
 ammonition_need_thousands: "[?daily_ammo_need|H1]§HK§!"
 potential_need_hundreds: "[?potential_need_h|H0]"
 potential_need_thousands: "[?potential_consumption_max|H1]§HK§!"
 ammunition_effects: "Lack of Ammunition"
 daily_ammunition_gain: "Daily Ammunition Gain"
 ammunition_factor: "Ammunition Gain Factor"
 ammunition_capacity_k: "Ammunition Capacity (K)"
 ammunition_plant: "Ammunition Plant"
 ammunition_plant_desc: "A Factory only dedicated to producing ammunition."
 ammunition_silo: "Ammunition Storage"
 ammunition_silo_desc: "A place to savely store ammunition in high quantities."
 ammunition_category: "Purchase Ammunition"
 ammunition_category_desc: "Purchase black market ammunition."
 buy_ammo_1000: "Buy 1,000 Rounds"
 buy_ammo_5000: "Buy 5,000 Rounds"
 buy_ammo_10000: "Buy 10,000 Rounds"
 buy_ammo_25000: "Buy 25,000 Rounds"
 buy_ammo_50000: "Buy 50,000 Rounds"
```

Ammo.1.d: "Fill our ammunition reserves with 100,000 rounds."

Ammo.1.a: "Fill our reserves!"

SEND_AMMO_LOC: "Send 50,000 rounds of ammo"

SEND_AMMO_RECIEVER_LOC: "You have been sent 50,000 rounds of ammo"

cost_0: "£treasury"

cost_0_blocked: "§§!"

cost_0_tooltip: "This purchase will cost §Hbillions§!"

ammo_eff: "Focus on Ammo Production"

ammo_eff_idea: "Focusing on Ammo Production"

ammo_eff_idea: "Trade military equipment production efficiency for ammo production efficiency."

ДОДАТОК В

Схема бази даних

```
generator client {
  provider = "prisma-client-js"
}
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}
model User {
  id         Int         @id @default(autoincrement())
  email      String      @unique
  password   String
  createdAt  DateTime    @default(now())
  translates Translate[]
}
model Translate {
  id             Int      @id @default(autoincrement())
  userId         Int
  originalFile   String
  translatedFile String
  originLang     String
  translatedLang String
  createdAt      DateTime @default(now())
  user User @relation(fields: [userId], references: [id], onDelete: Cascade)
  @@map("translates")
}
model PipelineFile {
  fileId       String           @id @map("file_id")
  status       String           @default("pending")
  totalLines   Int              @default(0) @map("total_lines")
  totalChunks  Int              @default(0) @map("total_chunks")
  originLang   String?          @map("origin_lang")
  targetLang   String?          @map("target_lang")
  createdAt    DateTime         @default(now()) @map("created_at")
  completedAt  DateTime?        @map("completed_at")
  error        String?
  progress     PipelineProgress?
  @@map("files")
```