КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота

на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

СИСТЕМА ПІДБОРУ ГРАВЦІВ В СЕСІЙНІЙ ОНЛАЙН ГРІ

Виконав студент 4-го курсу
Василь ПОГОРІЛИЙ _____
(підпис)

Науковий керівник:
кандидат технічних наук
Сергій КОНДРАТЮК _____
(підпис)

Засвідчую, що в цій роботі
немає запозичень з праць інших авторів без
відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на
засіданні кафедри теоретичної кібернетики
«____» _____ 2022 р.,
протокол № ___
Завідувач кафедри
доктор фіз.-мат. наук, професор
Юрій КРАК _____
(підпис)

Київ - 2023

РЕФЕРАТ

Обсяг роботи 49 сторінок, 29 ілюстрацій, 12 джерел посилань.

СЕРВЕРНИЙ ЗАСТОСУНОК, ОНЛАЙН-ГРА, СЕСІЯ, ПІДБІР ЗА ЯКІСНИМ ПОКАЗНИКОМ НАВИЧОК, JAVA, PYTHON, SPRING FRAMEWORK

Об’єктом роботи є серверний застосунок, який дає змогу, через конкретний api, надсилати на обробку гравця чи групу гравців, яким потрібно підібрати сеанс в грі спираючись на якісні показники їх навичок, та отримати в результаті список з n-ї кількості команд з рівним за навичками складом, для збалансованого та чесного підбору ігор.

Метою роботи є створення серверного застосунку та надання api для роботи з ним.

Результати роботи: вивчено та розроблено рішення, що дає змогу, використовуючи готовий api, користуватись логікою чесного та ефективного підбору ігор.

Розроблений серверний застосунок може бути використаний при розробці нових ігор, для впровадження логіки підбору гравців за їх навичками.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

API - прикладний програмний інтерфейс;

ML - машинне навчання;

БД - база даних;

IDE - інтегроване середовище розробки;

PvP - протистояння двох гравців між собою;

MM - зіставлення кількох гравців у команди для змагання

IoC - Inversion of Control

MVC - Model-View-Controller

Http - HyperText Transfer Protocol

URL - стандартизована адреса певного ресурсу

ВСТУП

Оцінка сучасного стану об’єкта розробки. Завдяки постійному росту індустрії відеоігор та зростанню онлайн-геймінгу, такі серверні застосунки стають все більш популярними та важливими для гравців та розробників ігор.

Оцінка залежить від кількох ключових факторів, таких як точність підбору гравців, ефективність роботи сервера, швидкість відповіді та загальна функціональність системи.

Щодо точності підбору гравців, цей серверний застосунок повинен мати високу рівність для забезпечення відповідного ММ. Це означає, що гравці зі схожими навичками та рівнем умінь повинні бути успішно згруповані разом для забезпечення збалансованої та конкурентної гри. Якщо система не здатна досягти цієї точності, можуть виникнути проблеми з нерівномірними матчами, довгими годинами очікування та загальним незадоволенням гравців.

Ефективність роботи сервера також є важливим аспектом оцінки. Серверний застосунок повинен бути здатним швидко обробляти велику кількість запитів, розраховувати матчі та забезпечувати стабільність під час навантаження. Якщо сервер не в змозі ефективно працювати зі зростаючою кількістю гравців, можуть виникати проблеми із затримками, лагами та непередбачуваним відмовами.

Швидкість відповіді сервера є ще одним ключовим фактором. Гравці очікують миттєвої відповіді від системи при пошуку сесії, якщо система працює повільно або виникають затримки, це може призвести до незадоволення та негативного досвіду користувачів.

Актуальність роботи та підстави для її виконання. Вирішення проблем підбору гравців у сесійних онлайн-іграх є дуже актуальним завданням. Довгий час очікування, нерівномірний підбір гравців та баги можуть негативно впливати на користувацький досвід і зменшувати задоволення від гри. Моя програма, яка пропонує універсальне рішення та використовує машинне навчання для покращення процесу підбору гравців, може вносити значний внесок у вирішення цих проблем.

Підстави для виконання роботи:

- Програма надає API для налаштування під конкретну гру, демонструє гнучкість та широкі можливості впровадження у різних іграх з ММ. Це забезпечує великий потенціал для застосування і популяризації системи підбору гравців у багатьох ігрових середовищах.
- Довгий час очікування, нерівномірний підбір гравців та баги є загальними проблемами у сесійних онлайн-іграх. Використання машинного навчання для обчислення рейтингу гравців на основі їх статистики з останніх ігрових сесій може допомогти покращити процес підбору і забезпечити більшу рівномірність та збалансованість.
- Потенційною користю мого застосунку є зменшення часу очікування гравців, поліпшення рівномірності підбору та зменшення кількості багів можуть позитивно вплинути на залучення нових гравців, збереження існуючих користувачів та підвищення загального задоволення від гри.

Мета й завдання роботи. Метою даної роботи є створення ефективної програми для підбору гравців в сесійних іграх.
Для досягнення цієї мети поставлено такі завдання:

- Дослідити існуючі застосунки для підбору гравців в сесійних іграх.
- Дослідити особливості існуючих систем.
- Розробити технічне завдання.
- Створити тестову рангову систему.
- Створити застосунок на мові програмування Java, який буде імплементувати логіку підбору сесій для онлайн ігор з гравців зі схожими якісними показниками навичок.
- Створити застосунок на мові програмування Python, який буде на основі минулих сесій, використовуючи ML, прогнозувати зміну якісного показника навичок у гравців для підвищення ефективності головного застосунку.

Об’єкт, методи й засоби дослідження або розроблення. Під час розробки застосунка було використано еволюційну модель, де кожна проміжна версія програми була оцінена з точки зору ефективності та коректності роботи.

Було створено окремий застосунок мовою програмування Python, що дало змогу прогнозувати зміну якісного показника навичок у гравців, спираючись на дані їх гри у минулих сесіях.

В якості інструменту створення програмного засобу було використано IntelliJ IDEA - інтегроване середовище розробки (IDE) мовою програмування Java.

Можливі сфери застосування. Можливості застосування системи підбору гравців в сесійних онлайн-іграх можуть бути різноманітні і розповсюджені у різних сферах. Ось декілька можливих сфер застосування:

- Комп'ютерні ігри: системи підбору гравців можуть бути використані в різних жанрах комп'ютерних ігор, включаючи шутери, стратегії, спортивні симулятори та багато інших. Покращений процес підбору гравців допоможе створити більш рівномірні та збалансовані команди, покращуючи загальний геймплей та задоволення від гри.
- Електронний спорт: у сфері електронного спорту, де змагання проходять в онлайн-середовищі, важливо мати ефективну систему підбору гравців. Рівномірні та конкурентоздатні матчі сприяють розвитку професійних гравців, підвищують якість змагань та сприяють розвитку спортивної сцени.
- Соціальні платформи: окрім традиційних комп'ютерних ігор, системи підбору гравців можуть бути використані на соціальних платформах, які пропонують ігровий контент. Наприклад, система може допомогти підбирати гравців для спільної гри або спілкування в онлайн-середовищі, сприяючи формуванню комунітетів та залученню користувачів.
- Бізнес-ігри та навчальні середовища: у сфері бізнес-ігор та навчальних середовищ системи підбору гравців можуть бути корисними для створення ефективних команд або груп для спільного розв'язання задач, сприяючи співпраці та взаємодії між учасниками.

1 ПРЕДМЕТНА ОБЛАСТЬ

1.1 Історія розвитку ММ

Історія розвитку ММ пов'язана з еволюцією комп'ютерних ігор та зміною вимог до онлайн-ігрового досвіду. У ранніх етапах онлайн-ігор, які з'явилися в 1970-х і 1980-х роках, матчмейкінг був простим, гравці тоді знаходились в тих самих локальних мережах або об'єднувалися за допомогою прямих підключень. У більшості випадків гравці самостійно організовували матчі і вибирали суперників.

У 1990-х роках з'явилися перші онлайн-платформи, такі як Battle.net від Blizzard Entertainment[10] та Zone.com[11] від Microsoft[12]. Ці платформи надавали ігровий сервіс, де гравці могли зареєструватися, створювати профілі та шукати суперників для гри. Матчмейкінг став більш структурованим, але все ще базувався на виборі суперників з доступних пропозицій.

У 2000-х роках виникла потреба в більш об'єктивному оцінюванні навичок гравців для покращення процесу ММ. Компанії почали впроваджувати рейтингові системи, які враховували результати ігор, статистику та інші фактори для визначення рівня навичок гравців. Це дозволило більш точно підібрати суперників, створюючи більш рівні матчі.

З розвитком технологій та аналізу даних, алгоритми ММ стали більш складними та удосконаленими. Вони використовувались для оцінки навичок гравців, аналізу статистики, прогнозування результатів та формування збалансованих команд. Машинне навчання та штучний інтелект також знайшли своє застосування в розвитку алгоритмів ММ для поліпшення якості та ефективності процесу.

Останнім трендом в розвитку ММ є персоналізований підхід до підбору гравців. За допомогою аналізу даних ігрового стилю, уподобань та

інших персональних факторів, системи ММ можуть намагатися створити ігри, які краще відповідають індивідуальним потребам та задоволенню гравців.

З розвитком інтернету, технологій та аналізу даних, матчмейкінг постійно еволюціонує для поліпшення ігрового досвіду гравців. Компанії продовжують розвивати нові методики та алгоритми, щоб забезпечити рівномірний та захоплюючий матчмейкінг у своїх онлайн-іграх.

1.2 Система рангів в іграх. Як вона впливає на підбір сесій

ММ (або matchmaking) - система що дозволяє підібрати гравцям таку ігрову сесію, в якій всі інші гравці будуть зі схожим рангом та зі схожими навичками. Матчмейкер старається створити команди або пари з гравців зі схожим рівнем рангу, щоб забезпечити збалансовані та цікаві поєдинки. Це може включати формування команд з однаковими сумарними рейтингами або забезпечення симетрії між командами. Існує багато алгоритмів за якими здійснюється підбір сесій, але всі вони працюють лише для конкретних видів ігор (PvP та інші). Слід також мати на увазі, що найчастіше гравці сесійних онлайн ігор хочуть грати зі своїми друзями, утворюючи свого роду команду, де кожен буде відповідати за конкретну роль, тому була додана можливість збирати кількох гравців у групи, та підбирати одну сесію для них. В такому випадку, для підбору буде використаний алгоритм, що повинен знайти наближений до середнього ранг цієї групи, та за ним підібрати для них спільну сесію.

Система рангів в іграх є одним з ключових елементів ММ, який впливає на процес підбору гравців. Вона дозволяє оцінити навички та рівень гри, забезпечуючи більш рівномірне та збалансоване формування команд або суперників. При оцінці якості гри система може включати такі фактори, як кількість перемог, результати проти сильних гравців,

ефективність гравця та багато інших параметрів. Оцінка навичок допомагає визначити рейтинг гравця та його місце в системі рангів.

Гравці в системі рангів зазвичай поділяються на різні ранги або діапазони, які відображають їх рівень навичок. Це можуть бути такі ранги, як "новачок", "срібний", "золотий", "елітний" і так далі. Розбиття на рівні допомагає групувати гравців зі схожим рівнем навичок, забезпечуючи більш рівні та конкурентоздатні матчі.

Система рангів часто пов'язана зі стимулюванням гравців до подальшого вдосконалення та гри. Зазвичай, вона передбачає прогресію через різні ранги або діапазони, де гравці можуть отримувати певні нагороди, досягаючи нових рівнів. Це може включати віртуальні нагороди, підвищення престижу або інші бонуси, які заохочують гравців до активності та залучення до гри.

1.3 Вибір засобів реалізації

1.3.1 Мова програмування Java

Java є однією з найулюбленіших мов програмування у світі, особливо в сфері розробки програмного забезпечення. Вона має безліч застосувань і використовується для створення різноманітних програм, включаючи веб-додатки, мобільні додатки, серверні додатки та багато інших. Java надає розробникам потужний набір інструментів, бібліотек і фреймворків, що сильно спрощують процес розробки програмного забезпечення. Використання Java дозволило мені створити надійний і високоефективний застосунок, що надає відкритий api онлайн-іграм для впровадження ММ.

1.3.2 Фреймворк Spring

Spring є одним з найпопулярніших фреймворків розробки програмного забезпечення на мові Java. Він надає розробникам потужні інструменти та функціонал для швидкої і ефективної розробки різноманітних додатків, включаючи веб-додатки, мікросервіси, додатки для обробки даних та інші.

Spring використовує принцип інверсії управління, що дозволяє розробникам зосередитись на бізнес-логіці своїх додатків, відокремлюючи їх від технічних деталей. Spring контейнер IoC керує залежностями між компонентами додатка та автоматично їх впроваджує.

Spring надає реалізацію патерну проектування MVC для розробки веб-додатків. Цей підхід дозволяє розбити додаток на модель (Model), представлення (View) та контролер (Controller), що полегшує розподіл обов'язків та підтримує шаблони проектування.

Завдяки модульній архітектурі, Spring дозволяє використовувати потрібні модулі для конкретного проекту, такі як Spring Security для забезпечення безпеки, Spring Data для роботи з базами даних та Spring Cloud для розробки мікросервісів. Використовуючи Spring Boot разом із Spring Data, розробники можуть легко створювати самостійні додатки, автоматизуючи багато рутинних задач розробки, таких як конфігурація та керування залежностями.

Для реалізації свого проекту, я обрав модуль Spring Boot в парі зі Spring Data, що дозволило легко і швидко створити серверний застосунок. Spring Boot надає простий та ефективний спосіб створення самостійних, готових до використання додатків. Основна перевага Spring Boot полягає в тому, що він автоматизує багато рутинних задач розробки, таких як конфігурація та управління залежностями. Це дозволяє швидше розпочати роботу над проектом, зосереджуючись на бізнес-логіці та

функціональності. Spring Boot також надає вбудовану підтримку для веб-додатків, що полегшує розробку API та інтеграцію з базами даних. Окрім окремого модуля, я також використовував багато інструментів, що містить основний фреймворк Spring. Слід зазначити, що проектуючи застосунок заздалегідь було продумано які модулі будуть використовуватись в майбутньому, наприклад, Spring Security.

Також Spring надає різноманітні інструменти для тестування додатків, включаючи підтримку юніт-тестування, інтеграційне тестування та автоматизоване тестування. Це дозволяє розробникам впевнено тестувати свої додатки та забезпечувати їх якість.

Загалом, Spring є потужним фреймворком, який полегшує розробку програмного забезпечення на Java. Він надає багато функціональних можливостей, що допомагають розробникам писати чистий, модульний та розширюваний код.

1.3.3 Система збірки Gradle

Gradle є потужною системою збірки, яка дозволяє автоматизувати процес збирання, тестування, розгортання та управління залежностями програмного забезпечення. Вона була спеціально розроблена для ефективної роботи з проектами будь-якої складності, незалежно від їх розміру чи типу.

Gradle використовує декларативну мову скриптів для опису залежностей, завдань та конфігурації проекту. Система надає можливість визначати власні завдання та розширювати функціональність системи збирання за допомогою плагінів. Це дозволяє налаштовувати процес збирання під свої потреби та використовувати додаткові інструменти та бібліотеки.

Фреймворк має потужну систему керування залежностями, що дозволяє легко визначати, імпортувати та управляти залежностями з репозиторіїв, таких як Maven або JCenter. Це полегшує використання сторонніх бібліотек та компонентів у проекті.

Система використовує паралельну збірку, що дозволяє ефективно використовувати ресурси комп'ютера та прискорювати процес збирання. Крім того, Gradle підтримує інкрементну збірку, що означає, що тільки змінені або необхідні для збірки компоненти будуть перезібрані, що зменшує загальний час збірки проекту.

Підсумовуючи, Gradle є потужною та гнучкою системою збирання, яка допомагає ефективно керувати залежностями, налаштувати процес збирання та автоматизувати дії в проекті. Використання Gradle разом з Java та Spring дозволяє розробляти та управляти своєю системою підбору гравців з великою ефективністю.

1.3.4 Мова програмування Python

Python є мовою програмування високого рівня, яка пропонує розробникам простоту та читабельність коду. Вона знайшла популярність серед як початківців, так і професійних розробників.

У Python використовуються відступи пробілами для визначення блоків коду, що сприяє зрозумілості та читабельності програм. Це робить Python дуже зручним для навчання та написання коду.

Мова поставляється з великою стандартною бібліотекою, яка включає в себе багато корисних модулів та інструментів для виконання різноманітних завдань, наприклад, включає роботу з рядками, файлами, мережами, базами даних, графікою та іншими функціями.

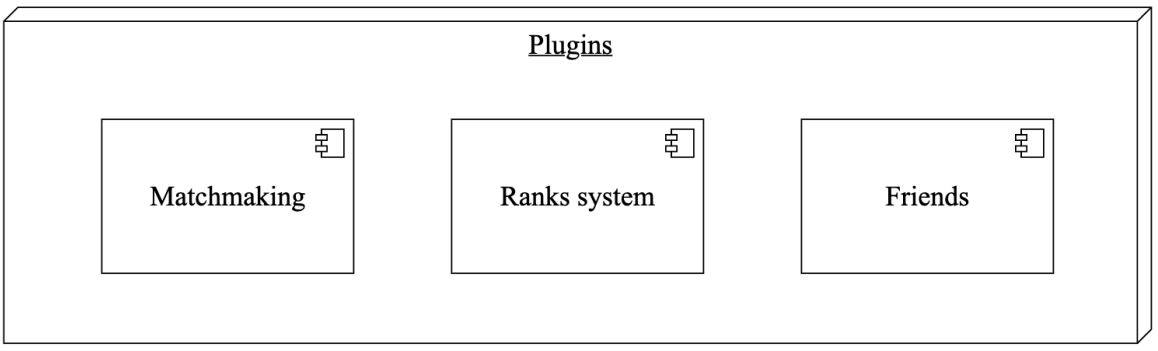
Одним із значних переваг Python є його підтримка різних парадигм програмування, таких як процедурне, об'єктно-орієнтоване та функціональне програмування. Ця мова стала популярною в багатьох галузях, включаючи веб-розробку, наукові дослідження, аналітику даних, машинне навчання та штучний інтелект. Тому, враховуючи все вищесказане, я вважаю, що вона є ідеальним вибором для розробки ще одного серверного застосунку, що буде відповідати за прогнозування рейтингу гравців.

2 ТЕОРЕТИЧНИЙ ОГЛЯД РОБОТИ ЗАСТОСУНКУ

2.1 Теоретичний огляд роботи серверного застосунку на Java

Проаналізувавши суть проблеми створення сервісу для підбору сесій, було вирішено архітектурно не обмежувати застосунок і залишити можливість створювати схожі сервіси, які б теж надавали відкритий арі і вирішували якусь проблему. Такі сервіси були зібрані у компонент, що має назву Plugins (рисунок 1), підпроекти якого являють собою реалізацію сервісів, які вирішують конкретні проблеми багатокористувацьких ігор. Прикладом таких сервісів може стати модуль Ranks system або модуль Friends (рисунок 1), що являють собою реалізацію рангової системи та системи друзів відповідно.

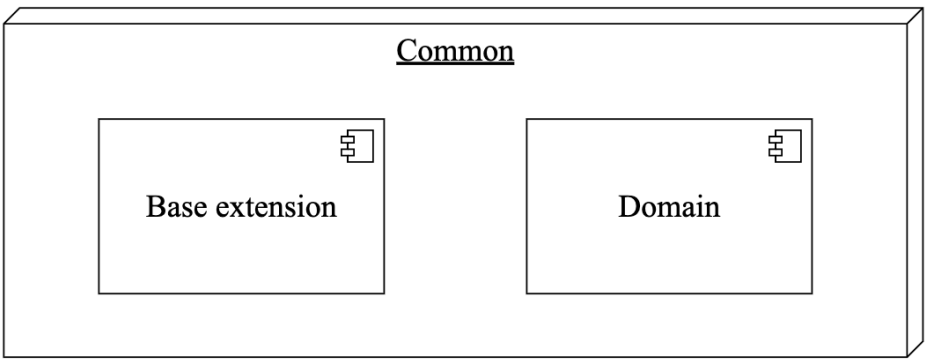
Кожен компонент з компоненту Plugins має набір базових залежностей, що дозволяє їм працювати незалежно один від одного, але крім базового набору, кожен компонент має свої унікальні залежності, притаманні його реалізації.



«Рисунок 1 – Структура компонента Plugins»

При розробці компонента Plugins виникла необхідність винесення спільної логіки його компонентів в окремий підпроект під назвою Base extension (рисунок 2), тепер програма зможе уникати великої кількості однотипного коду, кожен плагін може наслідувати його і доповнювати або використовувати спільну логіку основних сервісів. Цей підпроект є однією з складових компонента Common (рисунок 2), головною задачею якого є виокремити спільні сутності, сервіси з бізнес логікою, утилітарні сервіси, щоб полегшити розробку застосунка.

В майбутньому, при додаванні до проекту нових плагінів, всі вони повинні будуть унаслідувати і реалізувати від Base extension його основні інтерфейси, а також будуть мати доступ до всіх утилітарних сервісів які там містяться. Також нові плагіни будуть мати доступ до всіх сутностей які потрібні для роботи з базою даних.



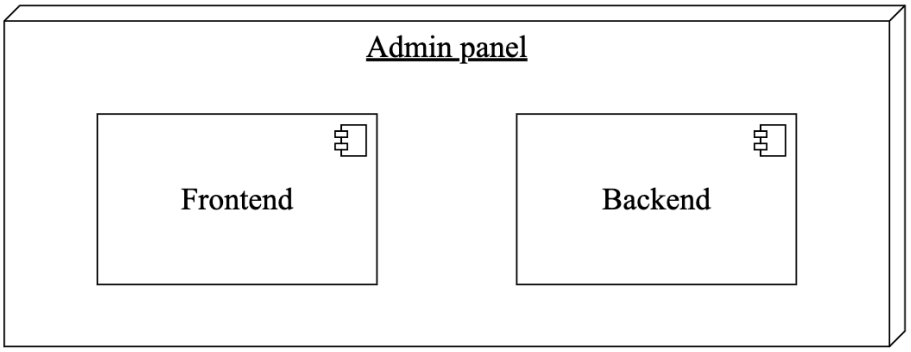
«Рисунок 2 – Структура компонента Common»

Однією зі складових компонента Common є підпроект Domain (рисунок 2), головною задачею якого є виокремлення сутностей які пов'язані з БД.

Для того щоб онлайн-ігри змогли використовувати api застосунку, спочатку необхідно здійснити першочергові налаштування:

- Додати назву гри, щоб застосунок зміг створити для неї унікальний id в БД.
- Якщо у грі присутня рангова система, то вказати її налаштування, для того щоб ММ адаптувався під неї.
- Вказати тип гри (PvP або інші).
- Вказати можливість групового пошуку сесій.
- За потреби додати адмінів, що можуть змінювати налаштування.

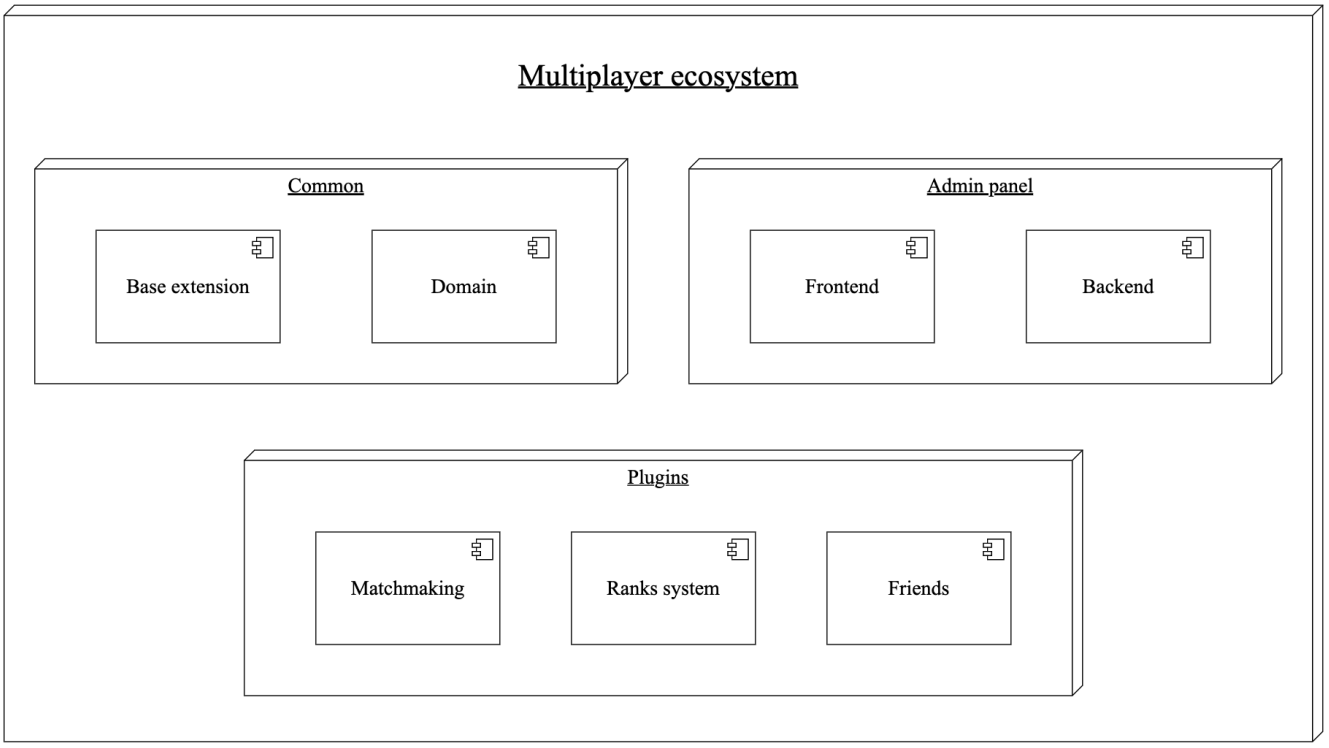
Ці налаштування, статистика пошуку, та інша інформація додається безпосередньо у компоненті під назвою Admin panel (рисунок 3). Це єдиний компонент серверного застосунку, що передбачає розробку візуальної складової (frontend). Головний розробник гри, або адмін якого поставили відповідальним за цю систему може у будь-який час гнучко змінити налаштування ММ для його гри.



«Рисунок 3 – Структура компонента Admin panel»

Оглянувши всі компоненти серверного застосунку і маючи уявлення як працює кожен з них, можна побачити як виглядає структура всього проекту в цілому (рисунок 4). Добре видно, що проект має мікросервісну архітектуру, кожен з плагінів запускається як окремий сервіс і всі вони спілкуються між собою відсилаючи Http запити.

Також варто зазначити, що на етапі проектування застосунку, заздалегідь було обрано таку архітектуру, що дасть змогу легко розширювати та підтримувати проект в майбутньому. Навіть якщо проект стане комерційним і над ним буде працювати багато розробників, це не стане проблемою, через його гнучку архітектуру, кожен розробник зможе незалежно розробляти нові, або підтримувати існуючі модулі.



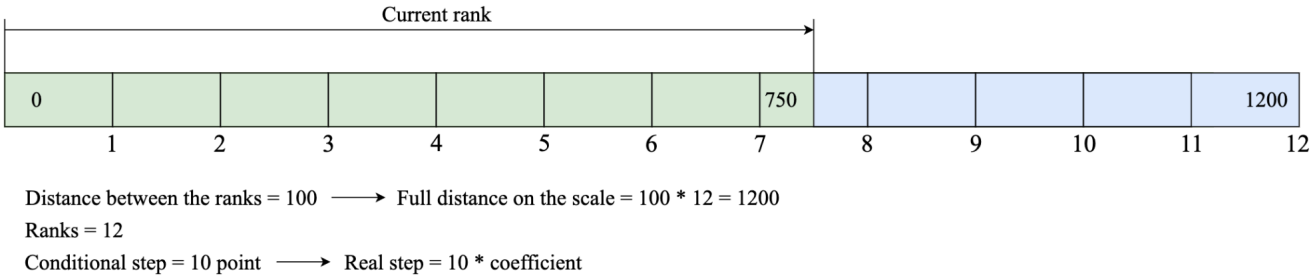
«Рисунок 4 – Структура серверного застосунку Multiplayer ecosystem»

2.2 Теоретичний огляд роботи серверного застосунку на Python

Головною задачею нашого застосунку є ефективний та справедливий підбір сесій для гравців, в цьому йому допомагає інший серверний застосунок написаний на мові програмування Python. Він взяв на себе роль “провісника”, використовуючи ML, базуючись на статистиці останніх сесій гравця, він прогнозує та записує в БД оновлене значення його рангу.

Після завершення сеансу на сервер приходить запит, з результатами всіх гравців, які перебували в цьому сеансі. Проаналізувавши всі значення, програма присвоює кожному гравцю оновлений коефіцієнт зміни рангу, що показує в яку сторону зміниться кількісний показник рангу.

Щоб зрозуміти суть роботи застосунку на Python, давайте розглянемо, як працює система рангів (рисунок 5).



«Рисунок 5 – Приклад рангової системи»

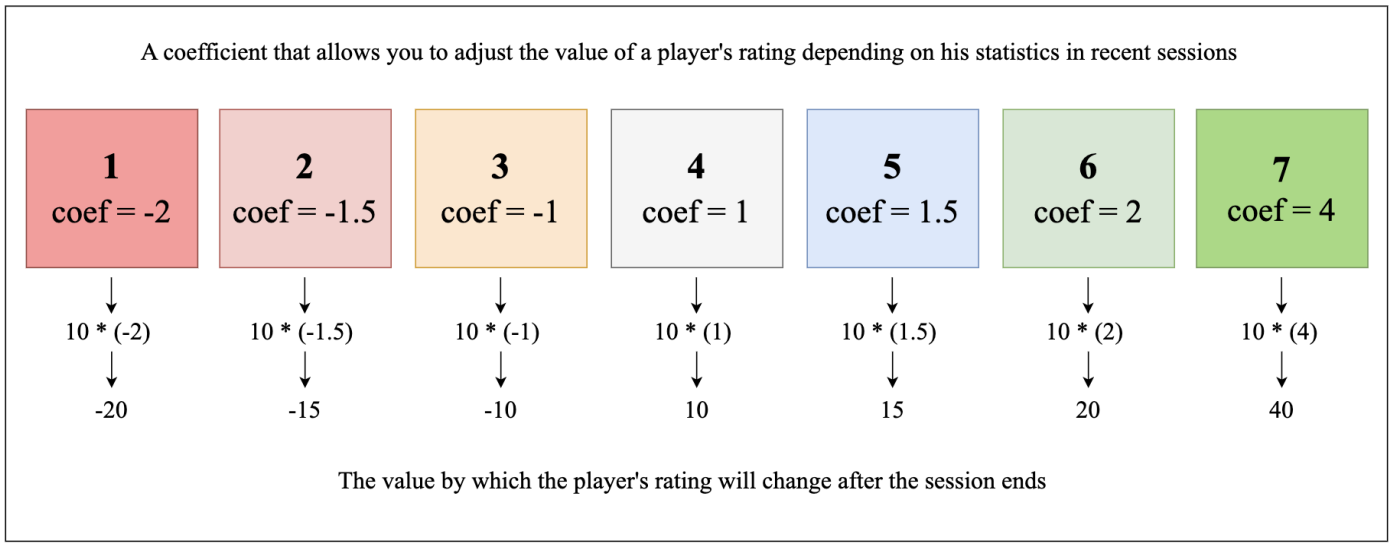
Всі значення які ми бачимо на рисунку (рисунок 5), будь то кількість рангів чи відстань між кожним рангом, підбираються і задаються унікально для кожної гри у компоненті Admin Panel їх командою розробників.

Проаналізуємо кожне з позначень наведених на рисунку (рисунок 5):

- Distance between the rank - це умовна відстань між кожним з рангів, яка характеризує кількість очок які потрібно набрати гравцю, щоб підвищити свій ранг.
- Ranks - кількість унікальних рангів, що дозволяє поділити гравців на задану кількість груп, відповідно до їх навичок.
- Full distance between the scale - значення, що показує скільки умовних ‘очок’ потрібно набрати гравцю від 1 до останнього рангу. Вона рахується, як добуток кількості рангів та відстані між рангами.
- Conditional step - умовний крок, що показує на яку величину не враховуючи коефіцієнт, буде змінюватись ранг гравця після завершення сесії.
- Real step - крок, що показує на яку величину буде змінюватись ранг гравця після завершення сесії. Він рахується як добуток умовного кроку та коефіцієнту.

Для маркування гравців в межах одного рангу, була розроблена система коефіцієнтів (рисунок 6). Кожен гравець, на основі своїх останніх n ігор, отримує від сервера один з 7 коефіцієнтів, який по завершенню кожного сеансу оновлюється.

Маючи поділ гравців на кожному з рангів, ми можемо змінювати їх якісні показники, для цього візьмемо значення, яке відповідає коефіцієнту гравця і помножимо його на умовний крок і отримаємо нове значення, що відповідає рівню гри гравця поміж інших гравців на даний момент. Нове значення буде записане у БД, та використане при підборі наступних сесій для гравця.



«Рисунок 6 – Розподіл коефіцієнтів»

Більш детально розглянемо наведений вище рисунок (рисунок 6), на ньому зображено 7 груп на які поділяються гравці в межах одного рангу, давайте розберемо кожну з них:

- 1 група - показує, що гравці вже не відповідають своїми навичками рівню інших гравців в межах свого рангу. Після завершення наступної сесії його ранг зменшиться на 20 одиниць.
- 2 група - показує, що гравці мають сильно гірші навички ніж інші гравці в межах свого рангу. Після завершення наступної сесії їх ранг зменшиться на 15 одиниць.
- 3 група - показує, що гравці мають трішки гірші навички ніж інші гравці в межах свого рангу. Після завершення наступної сесії їх ранг зменшиться на 10 одиниць.
- 4 група - показує, що гравці відповідають своїми навичками рівню інших гравців в межах свого рангу. Після завершення наступної сесії їх ранг збільшиться на 10 одиниць.

- 5 група - показує, що гравці мають трішки кращі навички ніж інші гравці в межах свого рангу. Після завершення наступної сесії їх ранг збільшиться на 15 одиниць.
- 6 група - показує, що гравці мають сильно кращі навички ніж інші гравці в межах свого рангу. Після завершення наступної сесії їх ранг збільшиться на 20 одиниць.
- 7 група - показує, що гравці відповідають своїми навичками гравцям наступного рангу. Після завершення наступної сесії їх ранг збільшиться на 40 одиниць.

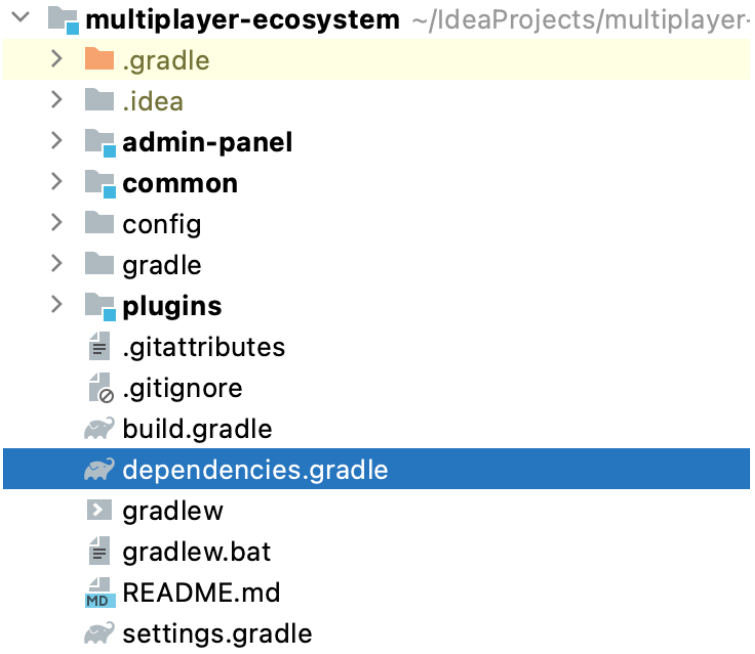
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ НА JAVA

3.1 Уніфікована система залежностей

Дуже часто, переглядаючи код своїх колег, чи open source проектів в мережі інтернет можна помітити, що найчастіше вони зіштовхуються з проблемою керування залежностями. В межах свого проекту я розробив уніфіковану систему додавання нових та керування старих залежностей.

Суть системи в тому, щоб забезпечити максимальний комфорт та уникнення помилок при додаванні нових залежностей у проект. Це досягається за рахунок того, що всі залежності які потрібні для роботи великого проекту містяться у одному файлі, що відповідно дає змогу легко керувати ними.

У якості файлу де будуть зберігатись усі залежності я обрав і створив у корені проекту файл dependencies.gradle (рисунок 7).



«Рисунок 7 – Файл зберігання залежностей»

Всередині файлу розміщені два масиви, в одному містяться версії залежностей, а в іншому їх шлях (рисунок 8).

```
ext {
    versions = [
            spring: '5.3.23',
            springBoot: '2.7.5',
            lombok: '1.18.24',
            log4j : '2.19.0',
            slf4j : '1.7.36',
            jackson: '2.15.1'
    ]

    dep = [
            spring: [
                    "org.springframework:spring-core:$versions.spring",
                    "org.springframework:spring-context:$versions.spring",
                    "org.springframework:spring-web:$versions.spring"
            ],
            springBoot: [
                    "org.springframework.boot:spring-boot-starter-web:$versions.springBoot",
                    "org.springframework.boot:spring-boot-starter-data-jpa:$versions.springBoot"
            ],
            springBootGradle: "org.springframework.boot:spring-boot-gradle-plugin:$versions.springBoot",

            lombok: "org.projectlombok:lombok:$versions.lombok",
            jackson: "com.fasterxml.jackson.core:jackson-databind:$versions.jackson"

    ]
}
```

«Рисунок 8 – Приклад уніфікованого зберігання залежностей у проекті»

Щоб всі модулі мали доступ до цього файлу, необхідно створити в корені проекту файл build.gradle, та збільшити видимість файлу для всіх підпроектів (рисунок 9).

Тепер щоб підключити в якийсь із модулів нову залежність, потрібно додати її до файлу dependencies.gradle, а потім підключити безпосередньо в build файлі потрібного модуля (рисунок 10). Видно, що тепер не потрібно буде згадувати в яких build файлах які залежності підключені, для того щоб видалити чи оновити її, достатньо просто вибрати необхідну залежність з масиву який знаходиться в dependencies.gradle, підключити її у потрібному файлі і при потребі внести які-небудь зміни.

```
allprojects {
    apply plugin: 'java'
    apply plugin: 'java-library'

    apply from: "${rootProject.projectDir}/dependencies.gradle"

    repositories {
        mavenCentral()
    }

    dependencies {
        implementation(dep.spring)
        implementation(dep.lombok)
        annotationProcessor(dep.lombok)
    }
}
```

«Рисунок 9 – Підключення файлу dependencies.gradle»

```
subprojects {
    dependencies {
        implementation project(':common')
        implementation(dep.springBoot)
        implementation(dep.jackson)
    }
}
```

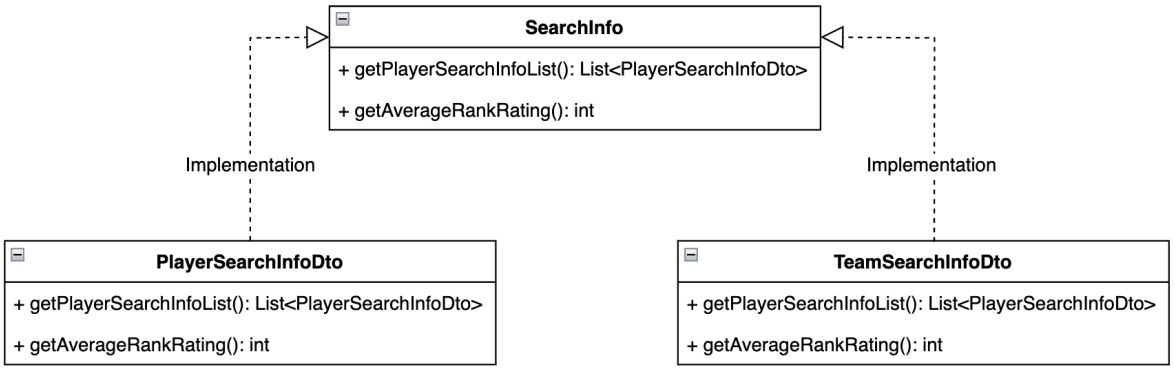
«Рисунок 10 – Підключення залежностей для модуля»

3.2 Архітектура підпроекту Matchmaking. Його програмна реалізація

3.2.1 Вирішення проблеми зберігання даних для пошуку сесії

Для початку розберемось із проблемою того, що ігри можуть відправляти на пошук сесії як одного гравця, так і групу гравців, відповідно, їх потрібно правильно зберегти, для подальшої легкої роботи з ними. Щоб вирішити це, було вирішено створити інтерфейс SearchInfo, який містив би інформацію про середній ранг гравця або групи гравців, яких він зберігає.

Для впровадження цієї логіки було створено ще два класи: PlayerSearchInfoDto та TeamSearchInfoDto, які в свою чергу наслідують інтерфейс SearchInfo, та імплементують його метод зі знаходження середнього рагу гравців (рисунок 11).



«Рисунок 11 – Діаграма класів для структури SearchInfo»

3.2.2 Обробка запитів онлайн-гри на додавання гравців у пошук

Вирішивши проблему зберігання різних груп гравців, можемо реалізувати систему, яка буде обробляти і зберігати їх в собі використовуючи структуру SearchInfo.

Для обробки вхідних запитів було створено SessionSearchController, що надає окремі методи для обробки одного гравця, або групи гравців. Кожен метод прив'язаний до конкретної URL адреси, що формується з @RequestMapping який вказується над назвою контролера (рисунок 12) і показує за яким шляхом можна до нього дістатись, та з більш конкретного @PostMapping або @GetMapping, що показують як дістатись до конкретного методу в класі (рисунок 13).

```
@Controller()
@RequiredArgsConstructor
@RequestMapping("/session")
public class SessionSearchController {
```

«Рисунок 12 – RequestMapping для SessionSearchController»

```
@PostMapping("/search/team")
public ResponseEntity<String> addTeamToSessionSearch(
        @RequestBody List<PlayerSessionSearchRequest> playerSessionSearchRequestList) {

    List<PlayerSearchInfoDto> teamSearchInfo = new LinkedList<>();

    playerSessionSearchRequestList.forEach(playerRequest -> teamSearchInfo.add(
            buildPlayerSearchInfo(
                    playerRequest.getPlayerId(),
                    playerRequest.getGameProviderKey()
            )
    ));

    TeamSearchInfoDto teamSearchInfoDto = TeamSearchInfoDto.builder()
            .teamSearchInfo(teamSearchInfo)
            .build();

    searchInfoHolder.addNewSearchInfo(teamSearchInfoDto);

    return new SuccessResponse<>();
}
```

«Рисунок 13 – PostMapping для методу додавання команд у пошук»

Наприклад, для того щоб надіслати запит для пошуку сесії одному гравцю, потрібно створити відповідний запит із заповненими даними, та відправити його по host://session/search/player. Відповідно для того щоб надіслати запит для групи гравців, потрібно створити відповідний запит із заповненими даними, та відправити його по host://session/search/team.

Для зберігання обробленої інформації було створено SearchInfoHolder, головною метою якого є забезпечити синхронізований доступ до збережених даних в конкурентному середовищі (рисунок 14).

```
@Holder
@RequiredArgsConstructor
public class SearchInfoHolder {

    @Getter
    private final List<SearchInfo> searchInfoList = new CopyOnWriteArrayList<>();

    public void addNewSearchInfo(SearchInfo searchInfo) {
        searchInfoList.add(searchInfo);
    }

    public void addAll(List<SearchInfo> infosToAdd) {
        searchInfoList.addAll(infosToAdd);
    }

    public void removeAll(List<SearchInfo> infosToRemove) {
        searchInfoList.removeAll(infosToRemove);
    }

}
```

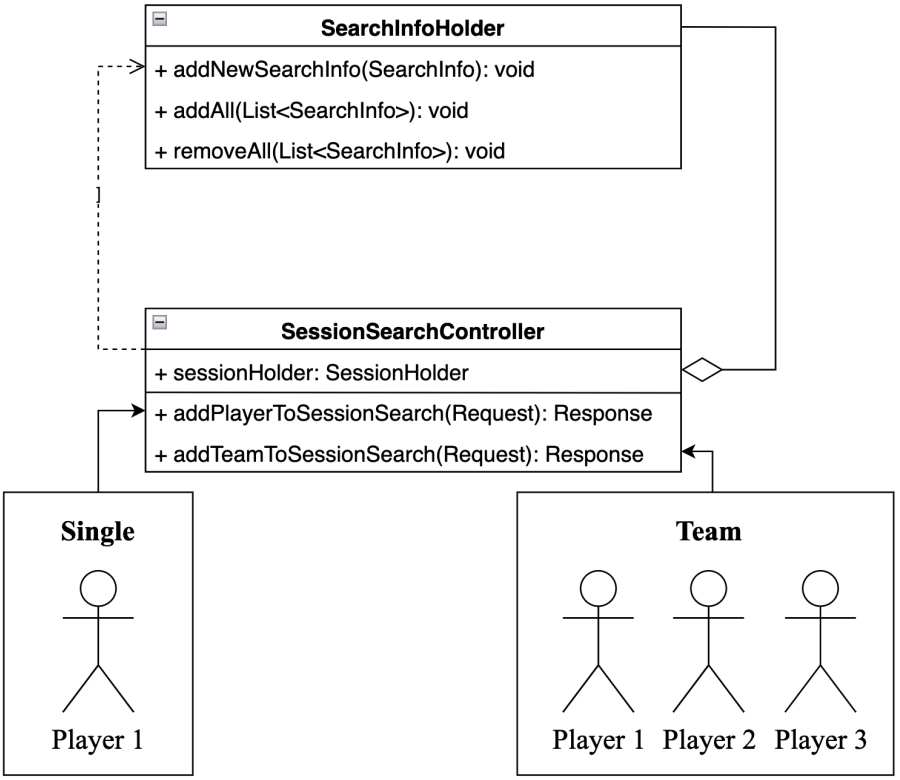
«Рисунок 14 – Релазація SearchInfoHolder»

Для досягнення синхронізованого доступу до збережених даних в конкурентному середовищі, було обрано потікобезпечну реалізацію інтерфейса List в Java - CopyOnWriteArrayList. Він забезпечує можливість

безпечного читання зі списку без потреби синхронізації, при цьому забезпечуючи консистентність даних навіть під час змін.

Основний принцип роботи CopyOnWriteArrayList полягає в тому, що кожна модифікація списку створює його копію з оновленнями, замість прямого змінювання оригінального списку. Це забезпечує потокобезпечність, оскільки кожен потік отримує свою власну копію для читання, а оригінальний список залишається незмінним.

Коли відбувається модифікація списку (наприклад, додавання або видалення елементів), створюється його копія зі змінами, а потім змінювані елементи копіюються в нову копію списку. Цей процес може бути витратним з точки зору пам'яті, особливо якщо список містить багато елементів або відбуваються часті модифікації. Однак, це забезпечує консистентність даних для потоків читання, незалежно від модифікацій, які відбуваються в інших потоках.



«Рисунок 15 – Загальна схема обробки запитів застосунком»

3.2.3 Обробка збережених даних. Формування сесій

Зберігши всі дані, нам тепер потрібно опрацьовувати їх. Застосунок повинен вміти раз в n секунд перевіряти SearchInfoHolder, на наявність достатньої кількості гравців для створення сесії і якщо потрібно, створити максимальну кількість сесій з наявний даних.

Для стабільного запуску завдань в окремих потоках, у Java наявний механізм під назвою ScheduledThreadPoolExecutor, головною задачею якого є запустити в одному із визначеної кількості потоків завдання, раз у заданий період часу.

Імплементуємо цей механізм у наш застосунок (рисунок 16).

```
@Service
public class TaskStarterService {

    private final ScheduledThreadPoolExecutor taskScheduler;

    public TaskStarterService(MatchmakingConfiguration matchmakingConfiguration) {
        this.taskScheduler = new ScheduledThreadPoolExecutor(
                matchmakingConfiguration.getDefaultThreadPoolSize()
        );
    }

    public void startRepeatableTask(Runnable task, int initialDelay, int period, TimeUnit unit) {
        scheduleAtFixedRate(task, initialDelay, period, unit);
    }

    private void scheduleAtFixedRate(Runnable task, int initialDelay, int period, TimeUnit unit) {
        taskScheduler.scheduleAtFixedRate(task, initialDelay, period, unit);
    }

}
```

«Рисунок 16 – Реалізація TaskStarterService»

Маючи механізм запуску завдань в окремих потоках з заданим періодом, потрібно імплементувати сервіс, що буде конфігурувати потрібну задачу, в нашому випадку це формування сесій (рисунок 17).

```java
@AllArgsConstructor
public class BuildSessionTaskService {

    private final TaskStarterService taskStarterService;
    private final MatchmakingService matchmakingService;
    private final MatchmakingConfiguration matchmakingConfiguration;

    public void startBuildSessionsTask() {
        taskStarterService.startRepeatableTask(new BuildSessionTask(matchmakingService),
                matchmakingConfiguration.getDefaultInitialTaskDelayInMillis(),
                matchmakingConfiguration.getDefaultGlobalTaskPeriodInMillis(),
                TimeUnit.MILLISECONDS
        );
    }

}
```

«Рисунок 17 – Реалізація BuildSessionTaskService»

Розглянемо детальніше рисунок (рисунок 17). Публічний метод startBuildSessionTask() викликається при старті застосунку і ініціює запуск сист ематичної збірки сесій із заданим періодом.

Для старту задачі нам потрібні такі параметри:

- Задача яка наслідує інтерфейс Runnable і в методі run() викликає відповідний метод в MatchamakingService.
- Значення що означає, з якою затримкою почнеться перше виконання завдання. Встановлюється для кожної гри індивідуально в Admin panel.
- Значення що означає, з яким періодом буде запускатись потрібна задача. Встановлюється для кожної гри індивідуально в Admin panel.
- Значення що показує, з якими розмірними одиницями буде працювати сервіс, що запускає завдання.

Реалізувавши і автоматизувавши запуск задачі з формування сесій, реалізуємо безпосередньо саме формування.

Для забезпечення коректної роботи сервісу з підбору сесій для гравців, розробимо і розглянемо безпосередньо саму логіку сервісу.

Метод що реалізує головну логіку сервісу повинен бути потікобезпечним, через те що він працює у конкурентному середовищі. Для цього, в першу чергу, створимо так званий “лок”, що блокує доступ до методу, поки якийсь із пулу потоків працює всередині нього (рисунок 18).

```
public Optional<List<SessionInfoDto>> buildSessions() {
    Lock lock = new ReentrantLock();

    try {
        lock.lock();
```

«Рисунок 18 – Створення локу»

Для початку нам потрібно перевірити, чи взагалі нам вистачає гравців для створення сесії, у разі позитивної відповіді повернемо ранг, де кількість гравців перевищує мінімально допустиму кількість потрібну для створення сесії (рисунок 19).

Розглянемо детальніше рисунок (рисунок 19). Послідовність виконання дій у методі:

1. Будуємо пріоритетну блокуючу чергу, яка буде містити в голові черги ранг гравців, яких найбільше в пошуках сесії, і відповідно, в кінці черги буде ранг, гравців якого найменше в пошуках сесії. Це зроблено для того, щоб максимально розвантажити сервер і обробляти спочатку найзагруженіші ділянки.
2. Для кожного рангу робимо перевірку, чи достатньо гравців цього рангу що є у пошуках сесії. При перевірці будемо враховувати і сусідні ранги, які будуть входити в межі погрішності, що задається при конфігуруванні застосунку в Admin panel.

```
private Optional<RankDto> getRankThatMatchEnoughPlayersToBuildSession() {
    Queue<RankDto> priorityRankQueue = buildPriorityQueueForRankProcessing();

    boolean isEnoughPlayersToBuildSession;
    while (!priorityRankQueue.isEmpty()) {
        RankDto rank = priorityRankQueue.poll();

        isEnoughPlayersToBuildSession = BigDecimal.valueOf(
                getNeededNumOfPlayersToBuildSession()
        ).compareTo(BigDecimal.valueOf(
                getNumOfPlayersForCurrentRankWithDivergence(rank))
        ) <= 0;

        if (isEnoughPlayersToBuildSession) {
            return Optional.of(rank);
        }
    }

    return Optional.empty();
}
```

«Рисунок 19 – Перевірка і повернення рангу з достатньою кількістю гравців для створення сесій»

Перевіряємо, чи отриманий результат не є пустим і за потреби приступаємо до формування сесій. Для цього спочатку знайдемо всіх гравців зі списку, що задовольняють найденому рангу зі встановленою в конфігурації погрішністю (рисунок 20).

```
if (!rankOptional.isPresent()) {
    return Optional.empty();
}

RankDto rank = rankOptional.get();
List<SearchInfo> playersToBuildSessions = findAllPlayersThatMatchCurrentRankWithDivergence(rank);
```

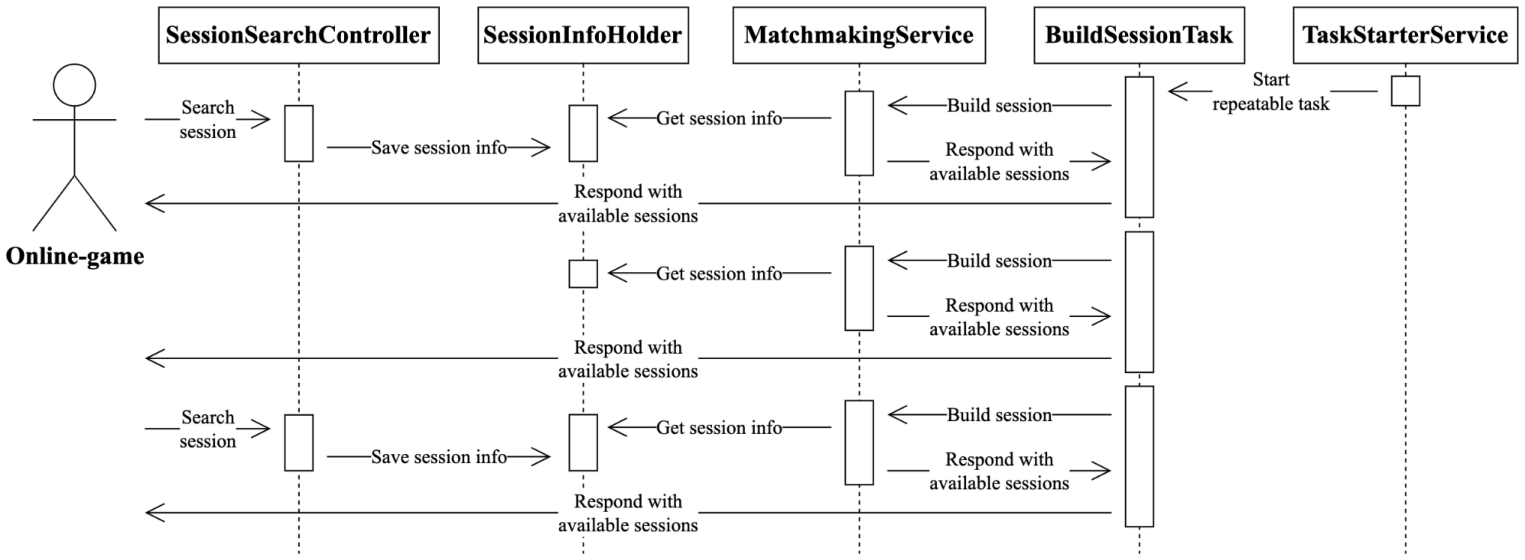
«Рисунок 20 – Обробка результату і знаходження гравців що задовольняють умову»

Для всіх знайдених гравців стараємось створити максимальну кількість сесій, а всіх тих хто залишився знову додаємо до пошуку (рисунок 21).

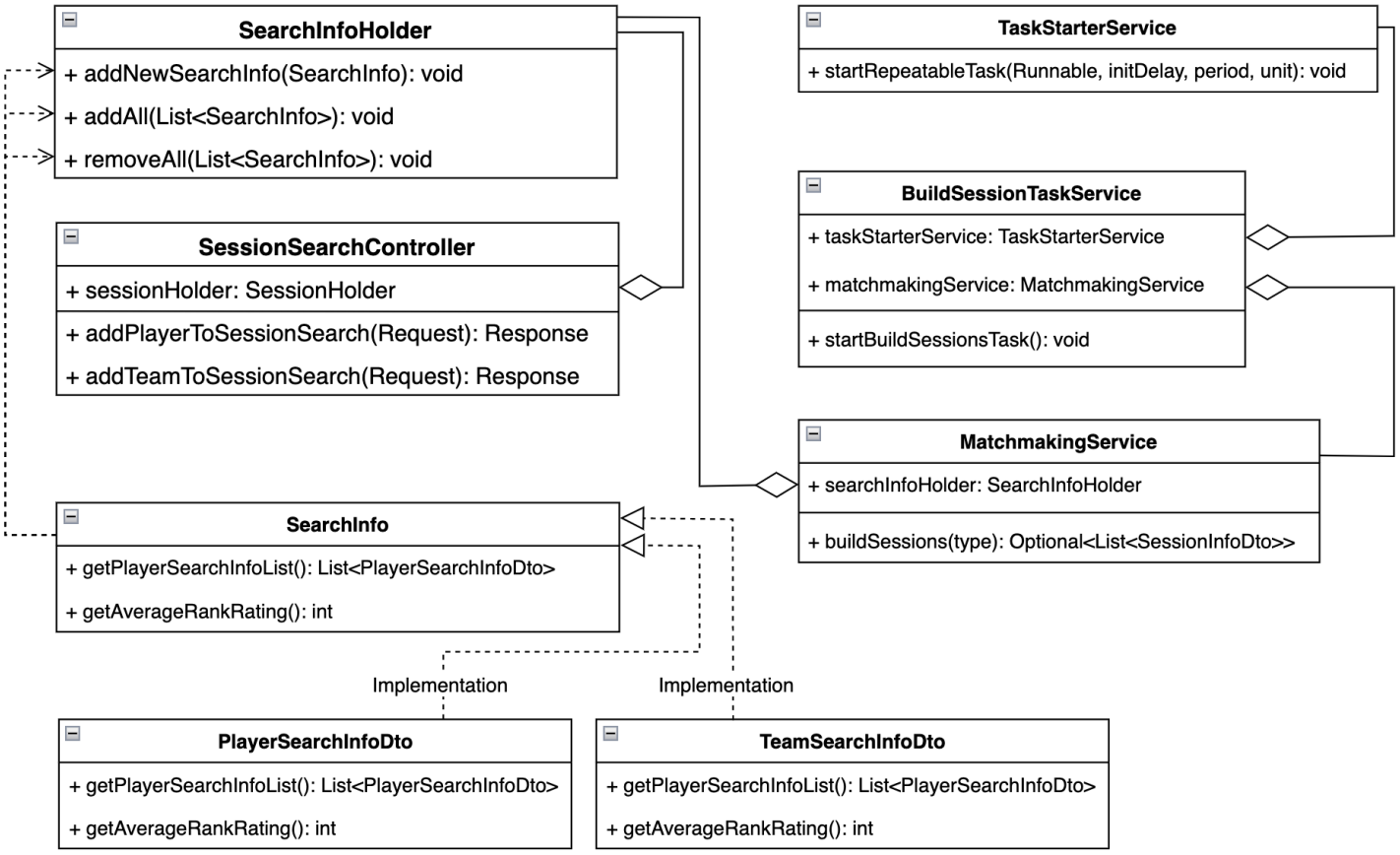
```
List<SessionInfoDto> readySessions = new LinkedList<>();
while (isCanBuildSession(playersToBuildSessions)) {
    readySessions.add(
            buildSession(playersToBuildSessions)
    );
}

if (!playersToBuildSessions.isEmpty()) {
    searchInfoHolder.addAll(playersToBuildSessions);
}
```

«Рисунок 21 – Створення максимальної кількості сесій для гравців»



«Рисунок 22 – Схема взаємодії всіх ключових сервісів в межах плагіну matchmaking. Sequence Diagram»



«Рисунок 23 – Діаграма класів для Matchmaking plugin»

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ НА PYTHON

Ця програма використовує модель лінійної регресії для прогнозування коефіцієнта на основі статистики гравця, обчислює зміну рангу на основі прогнозованого коефіцієнта та стандартної зміни рангу, а потім зберігає отримане значення в базі даних.

Коли ми говоримо про навчання моделі лінійної регресії, ми маємо на увазі процес визначення параметрів моделі таким чином, щоб вона могла найкращим чином підлаштуватися до навчальних даних.

Модель лінійної регресії припускає, що залежна змінна (у нашому випадку коефіцієнт) лінійно залежить від незалежних змінних (у нашому випадку статистика гравця). Таким чином, ми намагаємося знайти лінію, яка найкращим чином відображає цю залежність.

Процес навчання моделі лінійної регресії полягає в знаходженні оптимальних значень параметрів моделі, які мінімізують різницю між спостережуваними значеннями цільової змінної і прогнозованими значеннями, що видає модель.

Для початку імпортуємо необхідні бібліотеки: pandas і sklearn.linear_model.LinearRegression (рисунок 24). pandas використовується для роботи з даними, а LinearRegression - для навчання моделі лінійної регресії.

```
import pandas as pd
from sklearn.linear_model import LinearRegression
```

«Рисунок 24 – Імпорт потрібних бібліотек»

Тепер завантажуємо дані з файлу 'data.csv' за допомогою функції read_csv() з бібліотеки pandas. Дані зчитуються в об'єкт data (рисунок 25).

```
# Завантаження даних
data = pd.read_csv('data.csv')
```

«Рисунок 25 – Завантаження даних»

Розділяємо дані на ознаки та цільову змінну. Ознаками є статистика гравця (колонка 'Statistics' в data), яка зберігається в масиві X. Цільова змінна - коефіцієнт (колонка 'Coefficient' в data), який зберігається в масиві y (рисунок 26).

```
# Розділення даних на ознаки та цільову змінну
X = data['Statistics'].values.reshape(-1, 1)  # Ознаки - статистика гравця
y = data['Coefficient'].values  # Цільова змінна - коефіцієнт
```

«Рисунок 26 – Розділення даних на ознаки та цільову змінну»

Ініціалізуємо модель лінійної регресії за допомогою класу LinearRegression з sklearn. Потім навчаємо модель на навчальних даних X та y за допомогою методу fit() (рисунок 27).

```
# Навчання моделі лінійної регресії
model = LinearRegression()
model.fit(X, y)
```

«Рисунок 27 – Навчання моделі лінійної регресії»

Визначаємо функцію predict_coefficient(statistics), яка приймає значення статистики гравця та прогнозує коефіцієнт використовуючи

навчену модель. Використовується метод predict() моделі, щоб отримати прогнозоване значення (рисунок 28).

```
# Функція для прогнозування коефіцієнта на основі статистики гравця
1 usage
def predict_coefficient(statistics):
    coefficient = model.predict([[statistics]])[0]
    return coefficient
```

«Рисунок 28 – Функція прогнозування коефіцієнту»

Визначаємо функцію compute_rank_change(coefficient, standard_rank_change), яка приймає коефіцієнт та стандартну зміну рангу. Функція обчислює зміну рангу, перемножаючи коефіцієнт на стандартну зміну рангу (рисунок 29).

```
# Функція для обчислення зміни рангу на основі коефіцієнта
1 usage
def compute_rank_change(coefficient, standard_rank_change):
    rank_change = coefficient * standard_rank_change
    return rank_change
```

«Рисунок 29 – Функція обчислення зміни рангу»

Отже, ця програма використовує модель лінійної регресії для прогнозування коефіцієнта на основі статистики гравця, обчислює зміну рангу на основі прогнозованого коефіцієнта та стандартної зміни рангу, а потім зберігає отримане значення в базі даних.

ВИСНОВКИ

Метою моєї роботи було створення ефективної програми для підбору гравців в сесійних онлайн-іграх.
Для досягнення цієї мети було виконано такі завдання:

- Дослідив існуючі застосунки для підбору гравців в сесійних іграх.
- Дослідив особливості існуючих систем.
- Розробив технічне завдання.
- Створив тестову рангову систему.
- Створив застосунок на мові програмування Java, який імплементує логіку підбору сесій для онлайн ігор з гравців зі схожими якісними показниками навичок.
- Створив застосунок на мові програмування Python, який на основі минулих сесій, використовуючи ML, прогнозує зміну якісного показника навичок у гравців.

Провівши ряд досліджень існуючих застосунків для підбору сесій, виявивши всі їх недоліки, вдалося побудувати чітке технічне завдання, виконавши яке, отримали продукт, який здатний швидко та коректно виконувати завдання з підбору сесій в онлайн-іграх, надаючи їм для цього свій відкритий api.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. The Evolution of Online Gaming: https://play3r.net/news/the-evolution-of-online-gaming/
2. Online Gaming: A Successful Life? The Influence of Gaming on Development: https://www.researchgate.net/publication/345785658_Online_Gaming_A_Successful_Life_The_Influence_of_Gaming_on_Development
3. Online gaming: https://www.britannica.com/technology/Nintendo-console
4. A Brief History of Online Games: https://www.museumofplay.org/blog/a-brief-history-of-online-games/
5. Types of Computer Servers and How They Function: https://www.indeed.com/career-advice/career-development/types-of-servers
6. What is Java?: https://hackr.io/blog/what-is-java
7. What is Spring Framework?: https://www.marcobehler.com/guides/spring-framework
8. What is Gradle?: https://docs.gradle.org/current/userguide/what_is_gradle.html
9. What is Python?: https://www.infoworld.com/article/3204016/what-is-python-powerful-intuitive-programming.html
10. Blizzard Entertainment: https://www.blizzard.com/en-gb/
11. Zone.com: https://www.games.msn.com/en-us/home
12. Microsoft: https://www.microsoft.com/uk-ua

ДОДАТОК А

```
@Controller()
@RequiredArgsConstructor
@RequestMapping("/session")
public class SessionSearchController {

    private final SearchInfoHolder searchInfoHolder;
    private final PlayerRepository playerRepository;

    @PostMapping("/search/player")
    public ResponseEntity<String> addPlayerToSessionSearch(
            @RequestBody PlayerSessionSearchRequest playerSessionSearchRequest) {

        PlayerSearchInfoDto playerSearchInfoDto = buildPlayerSearchInfo(
                playerSessionSearchRequest.getPlayerId(),
                playerSessionSearchRequest.getGameProviderKey()
        );

        searchInfoHolder.addNewSearchInfo(playerSearchInfoDto);

        return new SuccessResponse<>();
    }

    @PostMapping("/search/team")
    public ResponseEntity<String> addTeamToSessionSearch(
        @RequestBody List<PlayerSessionSearchRequest> playerSessionSearchRequestList) {

        List<PlayerSearchInfoDto> teamSearchInfo = new LinkedList<>();

        playerSessionSearchRequestList.forEach(playerRequest -> teamSearchInfo.add(
                buildPlayerSearchInfo(
                        playerRequest.getPlayerId(),
                        playerRequest.getGameProviderKey()
                )
        ));

        TeamSearchInfoDto teamSearchInfoDto = TeamSearchInfoDto.builder()
                .teamSearchInfo(teamSearchInfo)
```

```
                .build();

        searchInfoHolder.addNewSearchInfo(teamSearchInfoDto);

        return new SuccessResponse<>();
    }

    private PlayerSearchInfoDto buildPlayerSearchInfo(Long playerId,
                                                      String gameProviderKey) {
        FullPlayerName fullPlayerName = FullPlayerName.builder()
                .playerId(playerId)
                .gameProviderKey(gameProviderKey)
                .build();

        PlayerEntity playerEntity = playerRepository
        .findByFullPlayerName(fullPlayerName);

        return PlayerSearchInfoDto.builder()
        .fullPlayerName(FullPlayerName.buildFromStringValue(
             playerEntity.getFullPlayerName()))
                .rank(playerEntity.getRank())
                .rankIndex(playerEntity.getRankIndex())
                .build();
    }

}
```

ДОДАТОК Б

```
@Service
@RequiredArgsConstructor
public class MatchmakingService {

    private final SearchInfoHolder searchInfoHolder;
    private final MatchmakingConfiguration matchmakingConfiguration;
    private final RanksInfo ranksInfo;

    public Optional<List<SessionInfoDto>> buildSessions() {
        Lock lock = new ReentrantLock();

        try {
            lock.lock();

            Optional<RankDto> rankOptional =
            getRankThatMatchEnoughPlayersToBuildSession();

            if (!rankOptional.isPresent()) {
                return Optional.empty();
            }

            RankDto rank = rankOptional.get();
            List<SearchInfo> playersToBuildSessions =
            findAllPlayersThatMatchCurrentRankWithDivergence(rank);

            List<SessionInfoDto> readySessions = new LinkedList<>();
            while (isCanBuildSession(playersToBuildSessions)) {
                readySessions.add(
                        buildSession(playersToBuildSessions)
                );
            }

            if (!playersToBuildSessions.isEmpty()) {
                searchInfoHolder.addAll(playersToBuildSessions);
            }

            return Optional.of(readySessions);
```

```
    } finally {
        lock.unlock();
    }
}

private boolean isCanBuildSession(List<SearchInfo> playersToBuildSessions) {
    return playersToBuildSessions.size() >= getNeededNumOfPlayersToBuildSession();
}

private SessionInfoDto buildSession(List<SearchInfo> playersToBuildSession) {
    List<TeamSessionInfoDto> teamSessionInfoDtoList = new LinkedList<>();

    while (isNeedToAddNewTeamToSession(teamSessionInfoDtoList)) {

        List<PlayerSessionInfoDto> playerSessionInfoDtoList =
        fillTeamWithNeededNumOfPlayers(
                playersToBuildSession
        );

        teamSessionInfoDtoList.add(TeamSessionInfoDto.builder()
                .playerSessionInfoDtoList(playerSessionInfoDtoList)
                .build());
    }

    return SessionInfoDto.builder()
            .teamSessionInfoDtoList(teamSessionInfoDtoList)
            .build();
}

private boolean isNeedToAddNewTeamToSession(List<TeamSessionInfoDto>
                                            teamSessionInfoDtoList) {
    return teamSessionInfoDtoList.size() != matchmakingConfiguration.getTeamNumber();
}

private List<PlayerSessionInfoDto> fillTeamWithNeededNumOfPlayers(List<SearchInfo>
                                                                  playersToBuildSession) {
    List<PlayerSessionInfoDto> playerSessionInfoDtoList = new LinkedList<>();

    for (SearchInfo searchInfo : playersToBuildSession) {

        List<PlayerSearchInfoDto> playerSearchInfoList =
```

```
            searchInfo.getPlayerSearchInfoList();
            if (playerSessionInfoDtoList.size() + playerSearchInfoList.size() <=
                    matchmakingConfiguration.getTeamPlayersNumber()) {

                playerSearchInfoList.forEach(playerSearchInfoDto ->
                    playerSessionInfoDtoList.add(
                        PlayerSessionInfoDto.builder()
                                .fullPlayerName(playerSearchInfoDto.getFullPlayerName())
                                .rank(playerSearchInfoDto.getRank())
                                .build())
                );

                playersToBuildSession.remove(searchInfo);
            }

            if (playerSessionInfoDtoList.size() ==
                   matchmakingConfiguration.getTeamPlayersNumber()) {
                break;
            }
        }

        return playerSessionInfoDtoList;
    }

    private List<SearchInfo> findAllPlayersThatMatchCurrentRankWithDivergence(
        RankDto rank) {
        List<SearchInfo> playersThatMatchCurrentRank =
             searchInfoHolder.getSearchInfoList().stream()
                .filter(searchInfo -> isMatchTheCurrentRankWithDivergence(
                         searchInfo, rank))
                .collect(Collectors.toList());

        searchInfoHolder.removeAll(playersThatMatchCurrentRank);

        return playersThatMatchCurrentRank;
    }

    private Optional<RankDto> getRankThatMatchEnoughPlayersToBuildSession() {
        Queue<RankDto> priorityRankQueue = buildPriorityQueueForRankProcessing();

        boolean isEnoughPlayersToBuildSession;
```

```
    while (!priorityRankQueue.isEmpty()) {
        RankDto rank = priorityRankQueue.poll();

        isEnoughPlayersToBuildSession = BigDecimal.valueOf(
                getNeededNumOfPlayersToBuildSession()
        ).compareTo(BigDecimal.valueOf(
                getNumOfPlayersForCurrentRankWithDivergence(rank))
        ) <= 0;

        if (isEnoughPlayersToBuildSession) {
            return Optional.of(rank);
        }
    }

    return Optional.empty();
}

private Queue<RankDto> buildPriorityQueueForRankProcessing() {
    List<RankDto> rankDtoList = ranksInfo.getRankDtoList();
    Queue<RankDto> priorityRankQueue = new LinkedBlockingQueue<>(rankDtoList.size());
    fillQueueAccordingToTheNumOfPlayersForEachRank(priorityRankQueue, rankDtoList);
    return priorityRankQueue;
}

private void fillQueueAccordingToTheNumOfPlayersForEachRank(
         Queue<RankDto> priorityRankQueue,
         List<RankDto> rankDtoList) {
    Map<RankDto, Integer> rankToPlayersCountMap = new HashMap<>();

    rankDtoList.forEach(rankDto -> {
        int numOfPlayersForCurrentRank = getNumOfPlayersForCurrentRank(rankDto);
        rankToPlayersCountMap.put(rankDto, numOfPlayersForCurrentRank);
    });

    rankToPlayersCountMap.entrySet()
            .stream()
            .sorted(Map.Entry.comparingByValue(Comparator.reverseOrder()))
            .forEach(entry -> priorityRankQueue.add(entry.getKey()));
}

private int getNumOfPlayersForCurrentRank(RankDto rank) {
```

```
    return searchInfoHolder.getSearchInfoList()
            .stream()
            .filter(searchInfo -> isMatchTheCurrentRank(searchInfo, rank))
            .map(SearchInfo::getPlayerSearchInfoList)
            .mapToInt(List::size)
            .sum();
}

private int getNumOfPlayersForCurrentRankWithDivergence(RankDto rank) {
    return searchInfoHolder.getSearchInfoList()
            .stream()
            .filter(searchInfo -> isMatchTheCurrentRankWithDivergence(
               searchInfo, rank))
            .map(SearchInfo::getPlayerSearchInfoList)
            .mapToInt(List::size)
            .sum();
}

private boolean isMatchTheCurrentRankWithDivergence(SearchInfo searchInfo,
                                                    RankDto rankDto) {
    boolean isBetweenRankBorder = isMatchTheCurrentRank(searchInfo, rankDto);

    int leftSideValueWithDivergence = rankDto.getLeftRankBorder() -
            matchmakingConfiguration.getDivergenceOfRankInSession();

    boolean isBetweenLeftBorderWithDivergence = leftSideValueWithDivergence > 0 &&
            MathUtil.isBetween(
                    searchInfo.getAverageRankRating(),
                    leftSideValueWithDivergence,
                    rankDto.getLeftRankBorder()
            );

    int rightSideValueWithDivergence = rankDto.getRightRankBorder() +
            matchmakingConfiguration.getDivergenceOfRankInSession();

    boolean isBetweenRightBorderWithDivergence = getMaxRating() >
            rightSideValueWithDivergence &&
            MathUtil.isBetween(
                    searchInfo.getAverageRankRating(),
                    rankDto.getRightRankBorder(),
                    rightSideValueWithDivergence
```

```
                );

        return isBetweenRankBorder ||
                isBetweenLeftBorderWithDivergence ||
                isBetweenRightBorderWithDivergence;
    }

    private boolean isMatchTheCurrentRank(SearchInfo searchInfo, RankDto rankDto) {
        return MathUtil.isBetween(
                searchInfo.getAverageRankRating(),
                rankDto.getLeftRankBorder(),
                rankDto.getRightRankBorder()
        );
    }

    private int getNeededNumOfPlayersToBuildSession() {
        return matchmakingConfiguration.getTeamPlayersNumber() *
                matchmakingConfiguration.getTeamNumber();
    }

    private int getMaxRating() {
        return matchmakingConfiguration.getRanksNumber() *
                matchmakingConfiguration.getDistanceBetweenRanks();
    }

}
```