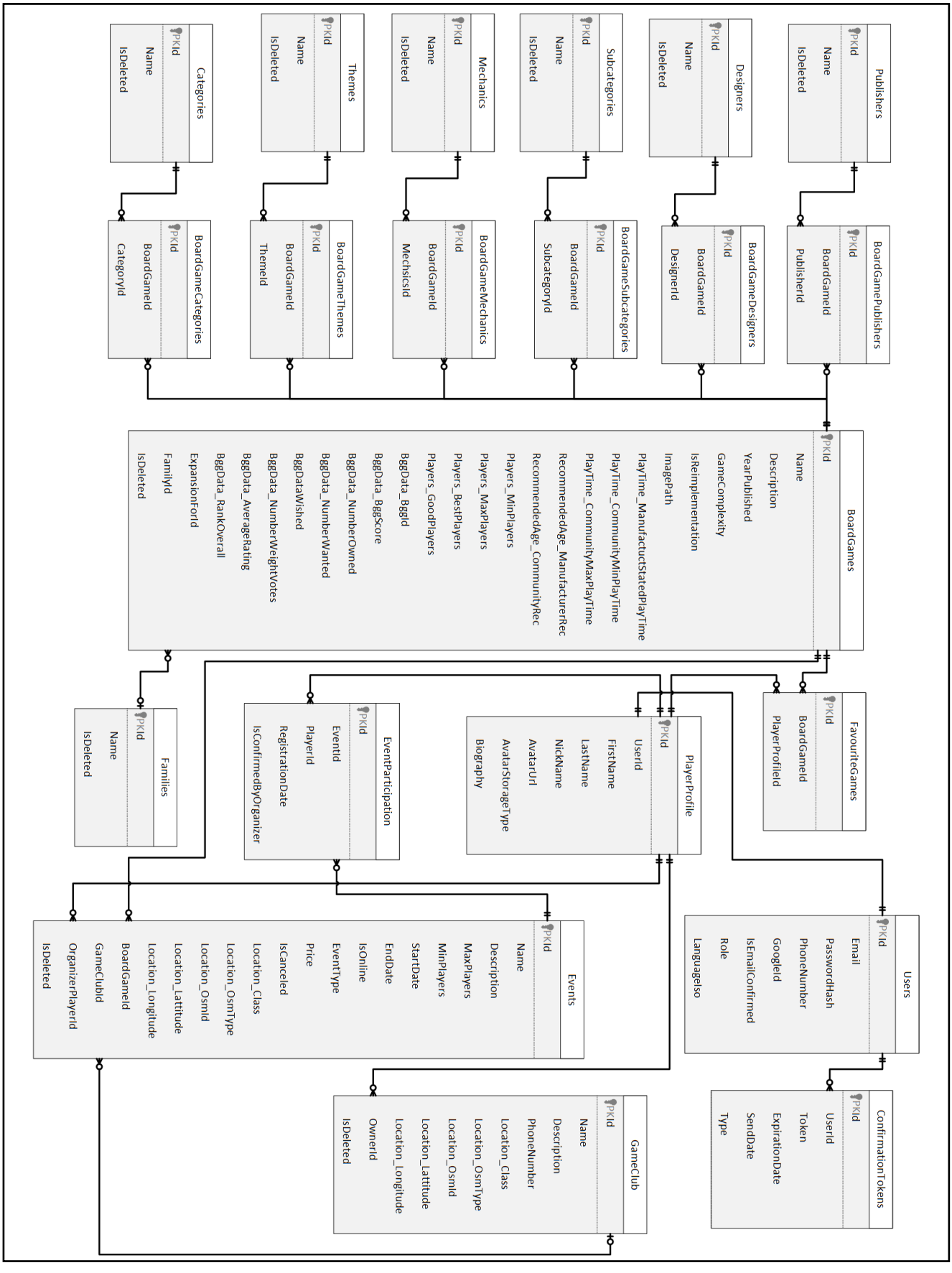
ДОДАТОК А

ER-діаграма у нотації Crow's Foot



ДОДАТОК Б

Фрагменти програмного коду серверної частини

Б.1 Код сервісу для імпорту даних про настільні ігри

```
     internal class BoardGamesCsvImportService :
IBoardGamesCsvImportService
     {
         private readonly CsvConfiguration csvReaderConfig =
new(CultureInfo.InvariantCulture)
         {
             Delimiter = ";",
             MissingFieldFound = null
         };
         private const string bggIdHeaderColumnName = "BGGId";
         private const string categoryHeaderColumnPrefix = "Cat:";
         private const string rankHeaderColumnName = "Rank:";
         private const string themeHeaderColumnPrefix = "Theme_";

         public BoardGamesCsvResultDto GetBoardGamesImportedData(Stream
csvStream)
         {
             csvStream.Position = 0;
             if (csvStream == null || csvStream.Length == 0)
                 throw new ArgumentException("CSV stream is empty.",
nameof(csvStream));


             using var reader = new StreamReader(csvStream);
             using var csv = new CsvReader(reader, csvReaderConfig);
             var records = new List<BoardGameCsvInputDto>();

             var categoriesProperties = typeof(BoardGameCsvInputDto)
                 .GetProperties()
                 .Where(prop => {
                     var attribute =
prop.GetCustomAttribute<NameAttribute>();
                     return attribute != null && attribute.Names.Any(n =>
n.StartsWith(categoryHeaderColumnPrefix,
StringComparison.OrdinalIgnoreCase));
                 })
                 .ToList();

             var rankProperties = typeof(BoardGameCsvInputDto)
                 .GetProperties()
                 .Where(prop => {
                     var attribute =
prop.GetCustomAttribute<NameAttribute>();
                     return attribute != null && attribute.Names.Any(n =>
n.StartsWith(rankHeaderColumnName, StringComparison.OrdinalIgnoreCase));
                 });

             csv.Read();

             if (!csv.ReadHeader() || csv.HeaderRecord == null)
```

```
                throw new InvalidOperationException("CSV header is
null.");

            var categories =
GetCategoriesFromImportedData(csv.HeaderRecord!);

            foreach (var record in
csv.GetRecords<BoardGameCsvInputDto>())
            {
                records.Add(record);
            }

            var families = records
                .Select(r => r.Family)
                .Distinct()
                .Where(f => !string.IsNullOrEmpty(f))
                .Select(f => new ClassifierCsvReturnDto(f!))
                .ToList();

            var boardGames = records.Select(r => new
BoardGameCsvReturnDto(
                r.BggId,
                r.Name,
                r.Description,
                r.YearPublished,
                r.GameWeight <= 1 ? 1 : r.GameWeight,
                r.BayesAvgRating,
                r.AvgRating,
                r.MinPlayers,
                r.MaxPlayers,
                r.ComAgeRec is not null?
(int)Math.Round(r.ComAgeRec.Value) : null,
                r.LanguageEase.HasValue &&
                    (int)Math.Round(r.LanguageEase.Value) >= 1 &&
                    (int)Math.Round(r.LanguageEase.Value) <= 5
                        ?
(LanguageDependence?)((int)Math.Round(r.LanguageEase.Value))
                        : null,
                r.BestPlayers,
                r.GoodPlayers,
                r.NumOwned,
                r.NumWant,
                r.NumWish,
                r.NumWeightVotes,
                r.MfgPlayTime,
                r.ComMinPlaytime,
                r.ComMaxPlaytime,
                r.MfgAgeRec,
                r.IsReimplementation,
                r.RankBoardGame,
                families.FirstOrDefault(f => f.Name == r.Family)?.Name,
                r.ImagePath)).OrderBy(r => r.BggId).ToList();

            var boardGameCategories = records
                .SelectMany(r =>
                {
                    return categoriesProperties
                    .Where(p =>
```

```
                    {
                        var gameInCategory = (bool)p.GetValue(r)!;
                        return gameInCategory;
                    })
                    .Select(p => new
BoardGameCategoryRelationCsvReturnDto(
                        r.BggId,
                        (categories.FirstOrDefault(
                            c => $"{categoryHeaderColumnPrefix}{c.Name}"
== (
                                p.GetCustomAttribute<NameAttribute>()?
                                .Names
                                .FirstOrDefault(n =>
n.StartsWith(categoryHeaderColumnPrefix,
StringComparison.OrdinalIgnoreCase)))?.Name) ?? "",
                        rankProperties.FirstOrDefault(
                            rankProp =>
rankProp.GetCustomAttribute<NameAttribute>()?
                                .Names.FirstOrDefault(n =>
n.StartsWith(rankHeaderColumnName,
StringComparison.OrdinalIgnoreCase))?.Replace(rankHeaderColumnName, "")
                                    ==
p.GetCustomAttribute<NameAttribute>()?.Names.FirstOrDefault(n =>
n.StartsWith(categoryHeaderColumnPrefix,
StringComparison.OrdinalIgnoreCase))?.Replace(categoryHeaderColumnPrefix,
""))
                                ?.GetValue(r) as int? ?? 0));
                })
                .ToList();

            return new BoardGamesCsvResultDto(boardGames, families,
categories, boardGameCategories);
        }

        public ClassifiersWithRelationsCsvResultDto
GetClassifiersWithRelationsImportedDataDefault(Stream csvStream)
        {
            return GetClassifiersWithRelationsImportedData(csvStream,
null);
        }

        public ClassifiersWithRelationsCsvResultDto
GetThemesImportedData(Stream csvStream)
        {
            return GetClassifiersWithRelationsImportedData(csvStream,
GetThemeNameFromHeader);
        }

        private ClassifiersWithRelationsCsvResultDto
GetClassifiersWithRelationsImportedData(
            Stream csvStream,
            Func<string, string>? classifierNameTransformSelector)
        {
            using var reader = new StreamReader(csvStream);
            using var csv = new CsvReader(reader, csvReaderConfig);

            csv.Read();
            csv.ReadHeader();
```

```
            if (csv.HeaderRecord == null)
                throw new InvalidOperationException("CSV header is
null.");

            var classifierRecords = csv.HeaderRecord.Where(r => r !=
bggIdHeaderColumnName).ToList();
            var classifiersDictionary = csv.HeaderRecord
                .Select((name, index) => new { name, index })
                .Where(r => r.name != bggIdHeaderColumnName)
                .ToDictionary(x => x.index, x => x.name);

            var classifierNames = classifierRecords;
            if (classifierNameTransformSelector != null)
            {
                classifierNames = classifierNames.Select(classifierName
=> classifierNameTransformSelector(classifierName)).ToList();
            }
            var classifiers = classifierNames.Select(p => new
ClassifierCsvReturnDto(p)).ToList();
            var boardGameWithClassifiersRelations = new
List<BoardGameClassifierRelationCsvReturnDto>();

            while (csv.Read())
            {
                int gameIdFieldIndex = csv.GetFieldIndex("BGGId");
                int gameId = csv.GetField<int>(gameIdFieldIndex);

                for (int i = 0; i < csv.HeaderRecord.Length; i++)
                {
                    if (i != gameIdFieldIndex && csv.GetField<int>(i) ==
1)
                    {
                        boardGameWithClassifiersRelations.Add(new(gameId,
classifiersDictionary[i]));
                    }
                }
            }
            return new(classifiers, boardGameWithClassifiersRelations);
        }

        private static List<ClassifierCsvReturnDto>
GetCategoriesFromImportedData(string[] headerRecords)
        {
            var categories = headerRecords.Where(headerRecords =>
headerRecords.StartsWith(categoryHeaderColumnPrefix)).ToList();
            return categories.Select(c => new
ClassifierCsvReturnDto(c.Split(":")[1])).ToList();
        }

        private static string GetThemeNameFromHeader(string header)
        {
            if (header.StartsWith(themeHeaderColumnPrefix))
                return header.Replace(themeHeaderColumnPrefix, "");

            return header;
        }
    }
```

Б.2 Код методів для алгоритму рекомендації настільних ігор

```
    public List<BoardGameRecommendationResultDto>
RecommendGames(List<int> favoriteGameIds, int topN = 10)
    {
        var favorites = _games.Where(g =>
favoriteGameIds.Contains(g.Id)).ToList();
        if (!favorites.Any()) return [];

        var userVector = AverageVector(favorites);

        var favouriteProfile = BuildFavoriteProfile(favorites);

        return _games
            .Where(g => !favoriteGameIds.Contains(g.Id))
            .Select(g =>
            {
                var vector = GameToVector(g);
                var similarity = CosineSimilarity(userVector, vector);

                var explanation = BuildExplanation(g, favouriteProfile);
                return new BoardGameRecommendationResultDto(
                    g.Id,
                    new BoardGameRecommendationExplanationDto(
                        similarity,
                        explanation.Categories,
                        explanation.Mechanics,
                        explanation.Themes,
                        explanation.Subcategories,
                        explanation.SharedFamily));
            })
            .OrderByDescending(r => r.Explanation.SimilarityScore)
            .Take(topN)
            .ToList();
    }

    private Vector<double> AverageVector(List<BoardGame> games) =>
        games.Select(GameToVector).Aggregate((a, b) => a + b) /
games.Count;

    private Vector<double> GameToVector(BoardGame g)
    {
        var v = new List<double>();

        double weightTags = 0.31;
        var tagVector = new List<double>();
        tagVector.AddRange(_allCategories.Select(id =>
g.BoardGameCategories.Any(c => c.CategoryId == id) ? 1.0 : 0.0));
        tagVector.AddRange(_allMechanics.Select(id =>
g.BoardGameMechanics.Any(m => m.MechanicsId == id) ? 1.0 : 0.0));
        tagVector.AddRange(_allThemes.Select(id =>
g.BoardGameThemes.Any(t => t.ThemeId == id) ? 1.0 : 0.0));
        tagVector.AddRange(_allSubcategories.Select(id =>
g.BoardGameSubcategories.Any(s => s.SubcategoryId == id) ? 1.0 : 0.0));
        var normalizedTagVector = NormalizeGroup(tagVector);
        v.AddRange(ScaleGroup(normalizedTagVector, weightTags));
```

```
            double weightAttributes = 0.12;
            var attributes = new List<double>
            {
                Normalize(g.GameComplexity, 1, 5),
                Normalize(g.RecommendedAge?.ManufacturerRecomended ?? 0, 3,
21),
                Normalize(g.PlayTime?.ManufacturerStatedPlayTime ?? 0, 5,
300),
                Normalize(g.Players?.MinPlayers ?? 1, 1, 20),
                Normalize(g.Players?.MaxPlayers ?? 1, 1, 20)
            };
            v.AddRange(ScaleGroup(NormalizeGroup(attributes),
weightAttributes));

            double weightRatings = 0.15;
            var ratings = new List<double>
            {
                Normalize(g.BggData?.BggScore ?? 0, 0, 10),
                Normalize(g.BggData?.AverageRating ?? 0, 0, 10)
            };
            v.AddRange(ScaleGroup(NormalizeGroup(ratings), weightRatings));

            double weightRank = 0.12;
            var rank = new List<double> {
NormalizeReverse((g.BggData?.RankOverall == null || g.BggData.RankOverall <
1) ? 30000 : g.BggData.RankOverall.Value, 1, 30000) };
            v.AddRange(ScaleGroup(rank, weightRank));

            double weightPopularity = 0.10;
            var popularity = new List<double> {
Normalize(g.BggData?.NumberOwned ?? 0, 0, 100000) };
            v.AddRange(ScaleGroup(popularity, weightPopularity));

            double weightYear = 0.12;
            var year = new List<double> { Normalize(g.YearPublished, 1950,
2025) };
            v.AddRange(ScaleGroup(year, weightYear));

            double weightFamily = 0.03;
            var family = new List<double>
            {
                g.FamilyId != null ? 1.0 : 0.0,
                g.IsReimplementation ? 1.0 : 0.0
            };
            v.AddRange(ScaleGroup(family, weightFamily));

            double weightLang = 0.05;
            var lang = new List<double> {
Normalize((double)g.LanguageDependence, 0, 5) };
            v.AddRange(ScaleGroup(lang, weightLang));

            return DenseVector.OfEnumerable(v);
        }
```

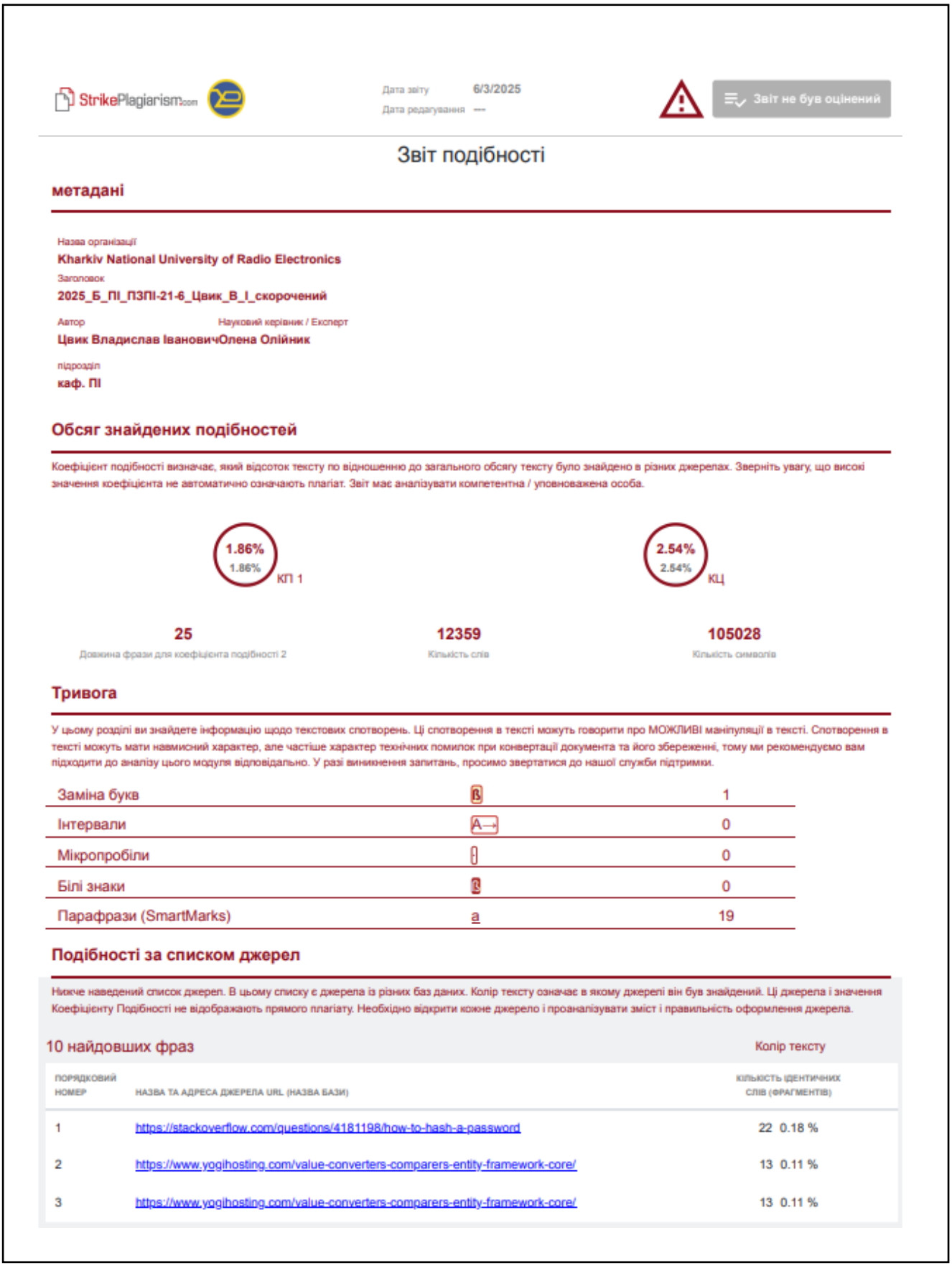
Б.3 Код сервісу для релевантного сортування подій

```
public class EventsRecommentaionService : IEventsRecommentaionService
{
    private const double W_ONLINE = 0.7;
    private const double W_DISTANCE = 1.7;
    private const double W_FAVORITE = 2.0;
    private const double W_CLUB = 1.5;
    private const double W_REGISTERED = 1.0;
    private const double W_NOGAME = 0.3;

    private readonly IEventsRepository _eventsRepository;

    public EventsRecommentaionService(IEventsRepository
eventsRepository)
    {
        _eventsRepository = eventsRepository;
    }

    public async Task<List<EventRecommendationReturnDto>>
GetRelevantEventsAsync(
        List<int> favouriteGames, LocationCoordinatesDto
playerLocation, int page, int take, CancellationToken cancellation)
    {
        var events = await
_eventsRepository.GetActiveEventsWithDistancesAsync(
            playerLocation.Longitude, playerLocation.Latitude,
cancellation);
        var scoredEvents = events.Select(e => new
        {
            Details = e,
            Score = CalculateEventRecommendationScore(e,
favouriteGames)
        });

        return scoredEvents
            .Select(e =>
                new EventRecommendationReturnDto(
                    e.Details.Event,
                    e.Details.BoardGameName,
                    e.Details.GameClubName,
                    e.Details.OrganizerFullName,
                    e.Details.OrganizerNickname,
                    e.Details.Distance,
                    e.Details.RegisteredPlayers,
                    e.Score))
            .OrderByDescending(e => e.RecommendationScore)
            .Skip((page - 1) * take)
            .Take(take)
            .ToList();
    }

    private double
CalculateEventRecommendationScore(EventWithDetailsDto details,
List<int> favouriteGames)
    {
        double score = 0.0;
```

```
        if (details.Event.IsOnline)
        {
            score += W_ONLINE;
        }

        if (details.Distance != null)
        {
            // f(dist) = 1 / (1 + dist), множимо на wDistance
            // Якщо dist = 0 => f(dist) = 1, чим більше dist, тим
менше вага.
            score += W_DISTANCE * (1.0 / (1 +
details.Distance.Value));
        }

        if (details.Event.BoardGameId != null)
        {
            var isFavourite =
favouriteGames.Contains(details.Event.BoardGameId.Value);

            if (isFavourite)
            {
                score += W_FAVORITE;
            }
        }
        else
        {
            score += W_NOGAME;
        }

        if (details.Event.GameCLubId != null)
        {
            score += W_CLUB;
        }

        // g(RegisteredPlayers) = ln(1 + RegisteredPlayers)
        var playersFactor = Math.Log(1 + details.RegisteredPlayers);
        score += W_REGISTERED * playersFactor;

        return score;
    }
}
```

ДОДАТОК В

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



StrikePlagiarism.com

Дата звіту 6/3/2025
Дата редагування —

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_Б_ПІ_ПЗПІ-21-6_Цвик_В_І_скорочений
Автор
Цвик Владислав Іванович
Науковий керівник / Експерт
Олена Олійник
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

1.86% 1.86% КП 1

2.54% 2.54% КЦ

25
Довжина фрази для коефіцієнта подібності 2

12359
Кількість слів

105028
Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		1
Інтервали		0
Мікропробіли		0
Білі знаки		0
Парафрази (SmartMarks)	a	19

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://stackoverflow.com/questions/4181198/how-to-hash-a-password	22 0.18 %
2	https://www.yogihosting.com/value-converters-comparers-entity-framework-core/	13 0.11 %
3	https://www.yogihosting.com/value-converters-comparers-entity-framework-core/	13 0.11 %

ДОДАТОК Г

Слайди презентації

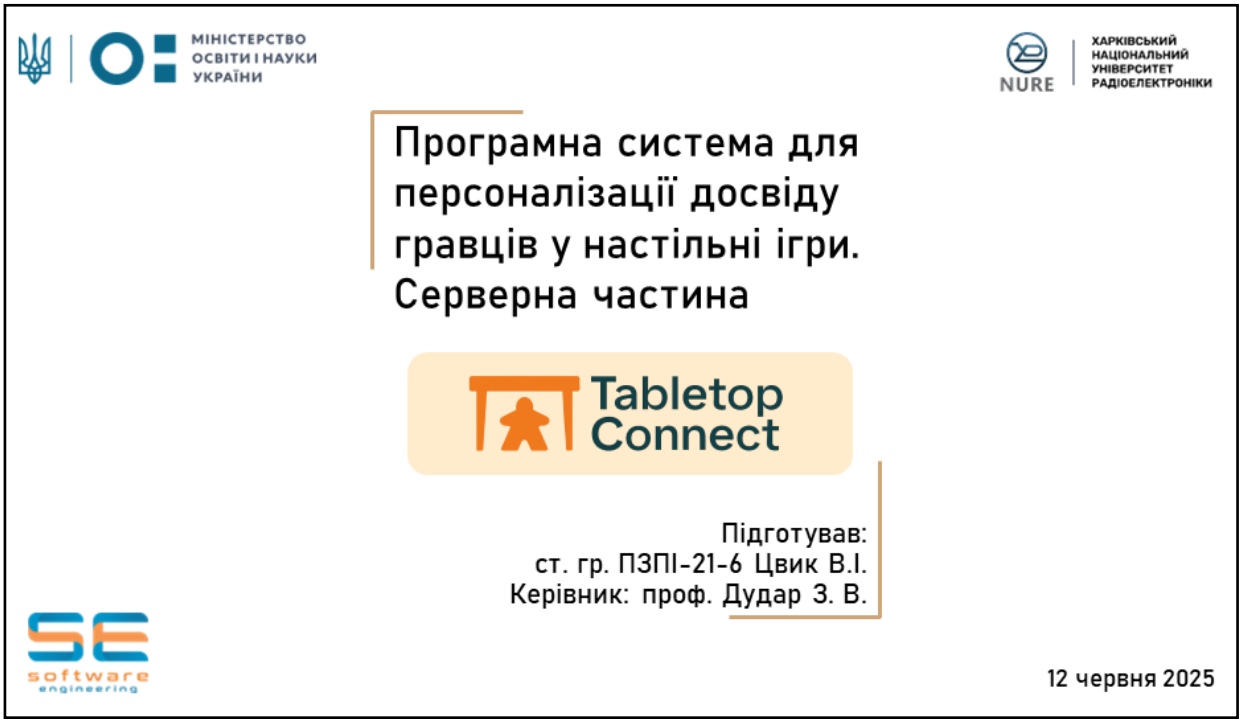


Рисунок Г.1 – Слайд №1 (рисунок виконано самостійно)

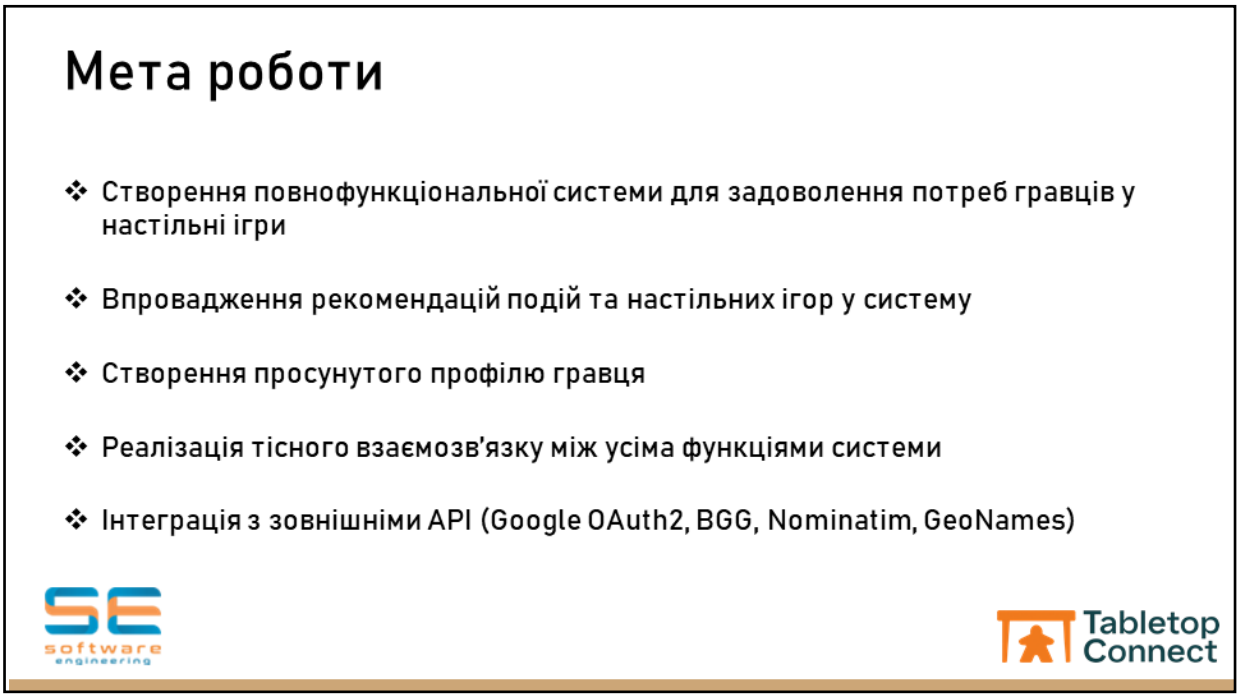


Рисунок Г.2 – Слайд №2 (рисунок виконано самостійно)

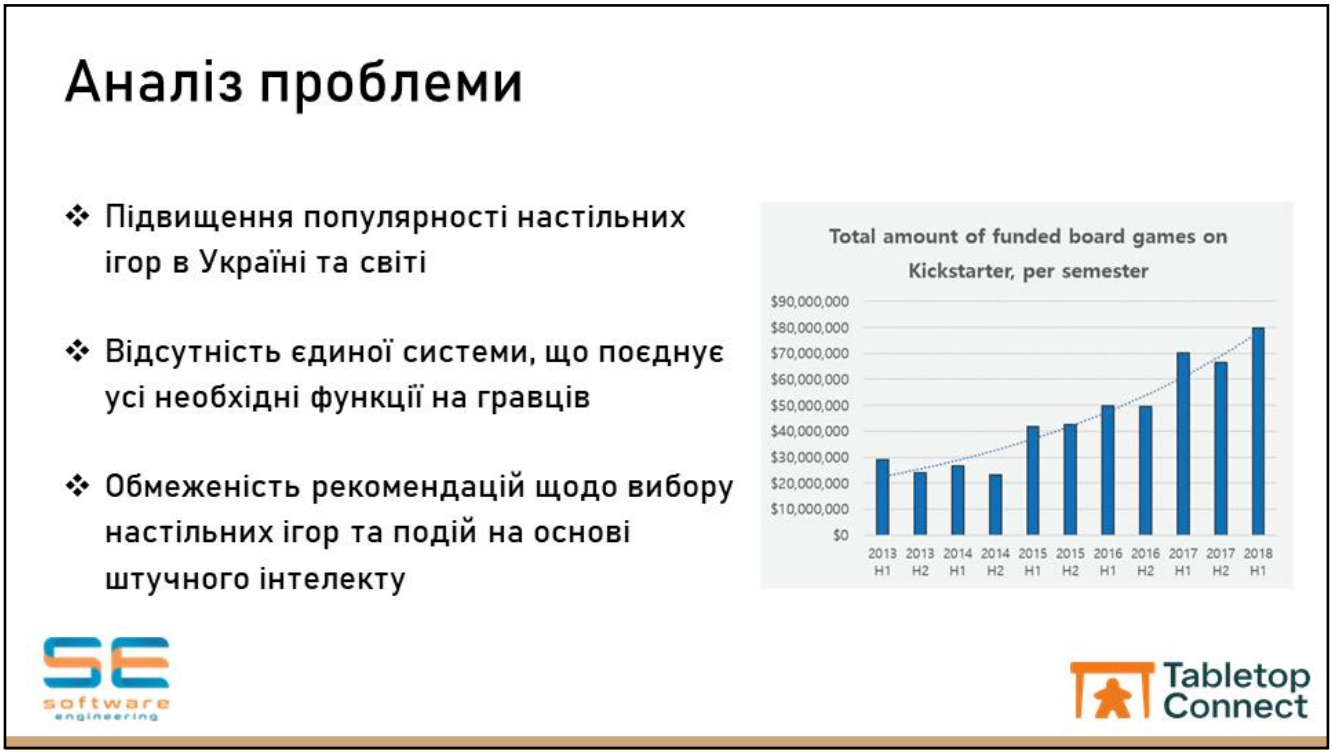


Рисунок Г.3 – Слайд №3 (рисунок виконано самостійно)

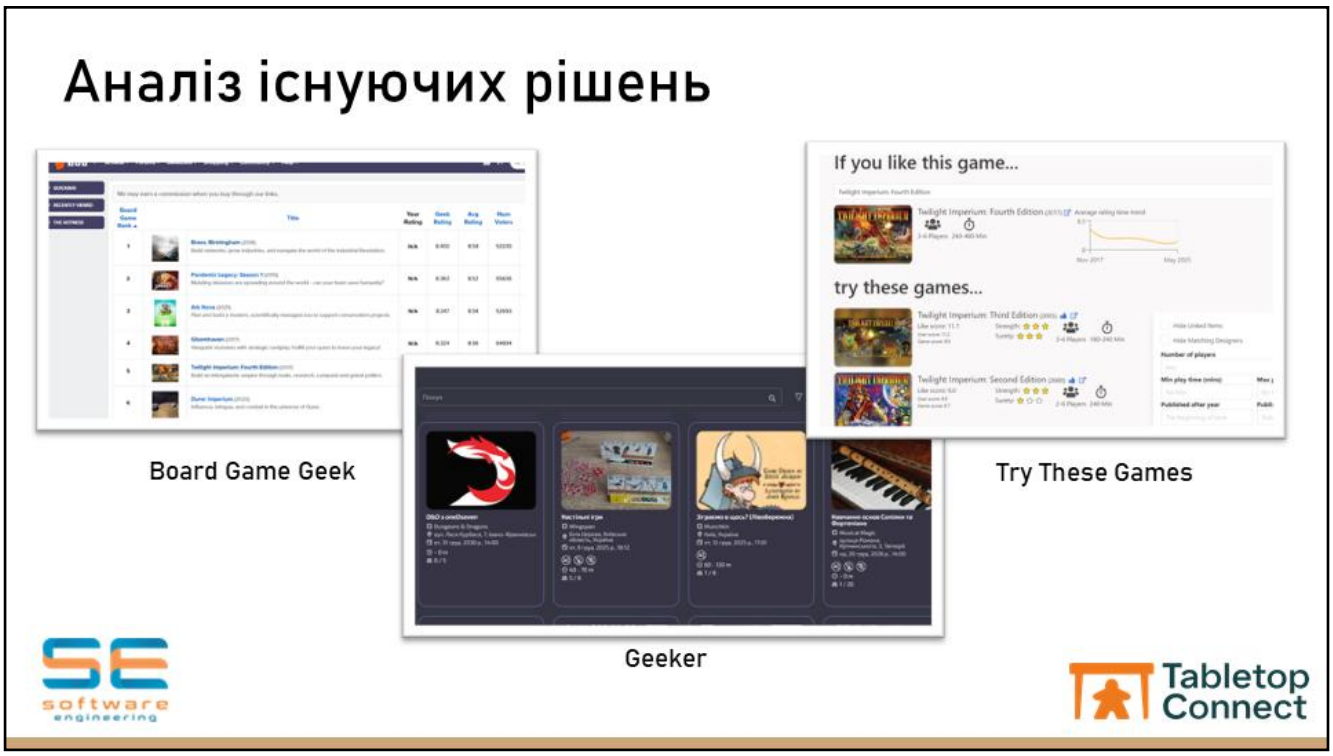


Рисунок Г.4 – Слайд №4 (рисунок виконано самостійно)

Рисунок Г.5 – Слайд №5 (рисунок виконано самостійно)

Рисунок Г.6 – Слайд №6 (рисунок виконано самостійно)

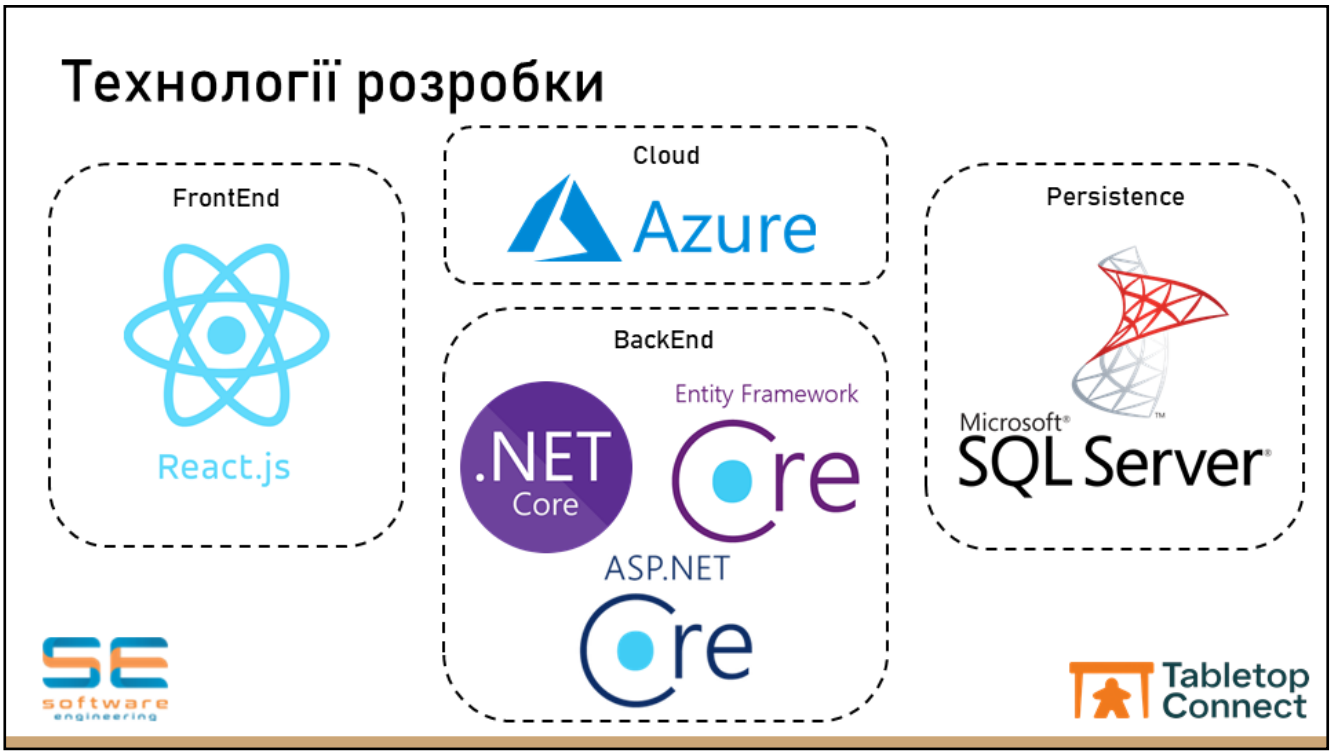


Рисунок Г.7 – Слайд №7 (рисунок виконано самостійно)

Рисунок Г.8 – Слайд №8 (рисунок виконано самостійно)

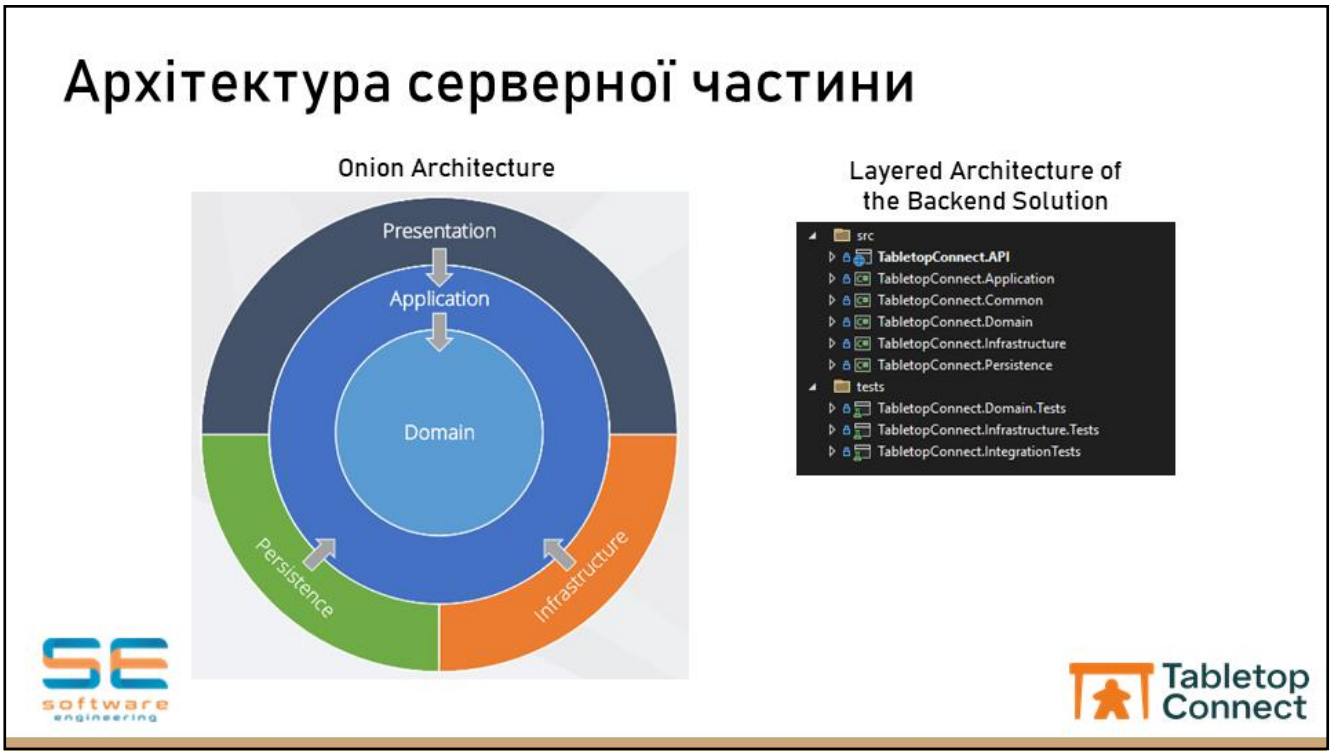


Рисунок Г.9 – Слайд №9 (рисунок виконано самостійно)

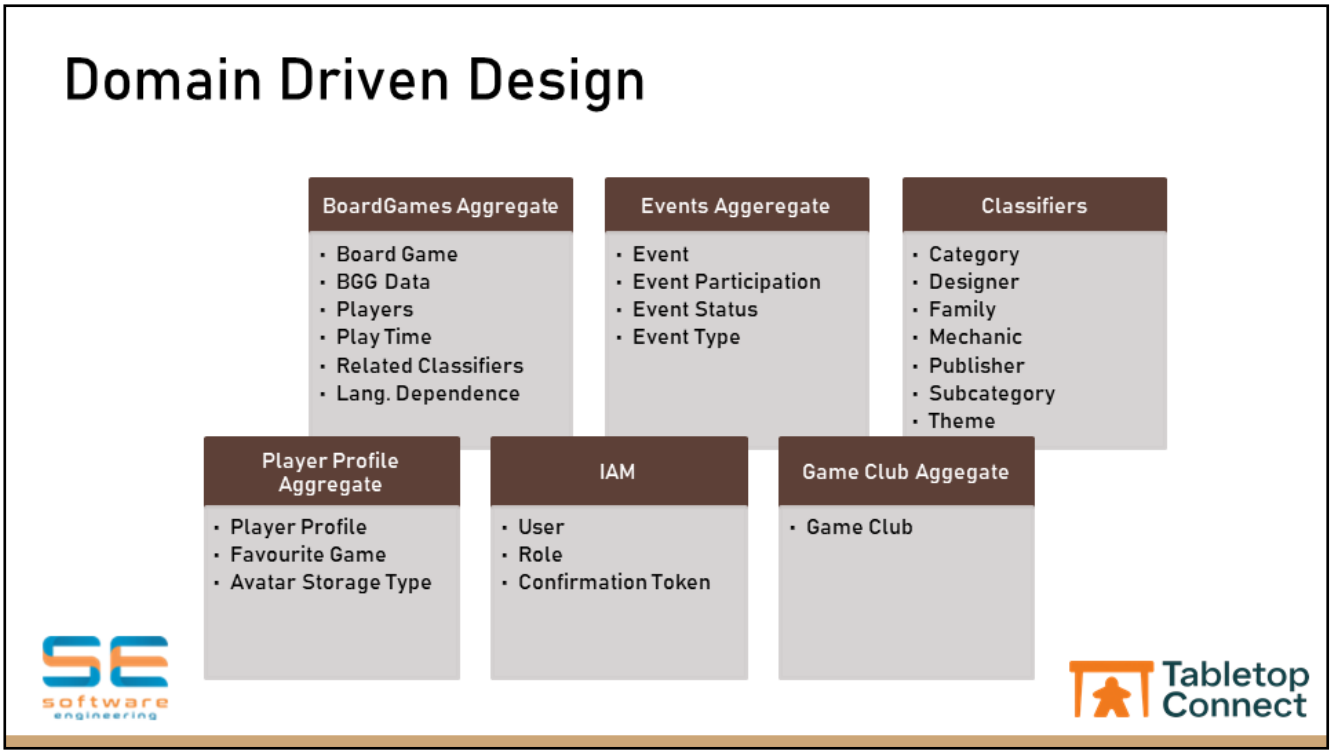


Рисунок Г.10 – Слайд №10 (рисунок виконано самостійно)

Рисунок Г.11 – Слайд №11 (рисунок виконано самостійно)

Рисунок Г.12 – Слайд №12 (рисунок виконано самостійно)

Алгоритм релевантного сортування подій

Фактор	Позначення	Вага	Розрахунок	Призначення
Онлайн подія	W_{online}	0.7	Додається, якщо подія онлайн	Онлайн-події гнучкіші у доступі.
Відстань до події	$W_{distance}$	1.7	1 / (1 + distance)	Чим ближча подія, тим вищий бал.
Улюблена настільна гра	W_{fav}	2.0	Якщо гра є в списку улюблених	Перевага для улюблених ігор.
Без гри (відкрита подія)	W_{nogame}	0.3	Якщо гра не вказана	Низький бал, бо невідомо, чи буде цікаво.
Прив'язка до ігрового клубу	W_{club}	1.5	Додається, якщо є клуб	Клубні події вважаються якіснішими.
Кількість зареєстрованих гравців	W_{reg}	1.0	ln(1 + N)	Більше людей — потенційно цікавіше.

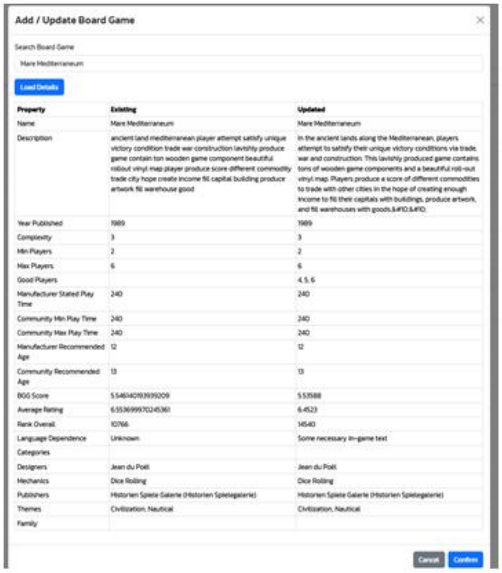
Events relevant soring method

```
private double CalculateEventRecommendationScore(EventWithDetailsDto details, List<int> favouriteGames)
{
    double score = 0.0;

    if (details.Event.IsOnline)
    {
        score += W_ONLINE;
    }

    if (details.Distance != null)
    {
        // f(dist) = 1 / (1 + dist), монотонно на зДistance
        // Якщо dist = 0 => f(dist) = 1, чим більше dist, тим менше вага.
        score += W_DISTANCE * (1.0 / (1 + details.Distance.Value));
    }

    if (details.Event.BoardGameId != null)
    {
        var isFavourite = favouriteGames.Contains(details.Event.BoardGameId.Value);

        if (isFavourite)
        {
            score += W_FAVORITE;
        }
    }
    else
    {
        score += W_NOGAME;
    }

    if (details.Event.GameClubId != null)
    {
        score += W_CLUB;
    }

    // g(RegisteredPlayers) = ln(1 + RegisteredPlayers)
    var playersFactor = Math.Log(1 + details.RegisteredPlayers);
    score += W_REGISTERED * playersFactor;

    return score;
}
```

$$score = W_{online} + W_{distance} + W_{fav} + W_{nogame} + W_{club} + W_{reg}$$

Рисунок Г.13 – Слайд №13 (рисунок виконано самостійно)

Інтеграції: BoardGameGeekAPI



BGG XML HTTP Client

```
public class BggXmlClient : IBggXmlClient
{
    private readonly HttpClient _httpClient;
    private readonly string _baseUrl = "https://boardgamegeek.com/xmlapi2";

    public BggXmlClient(HttpClient httpClient)
    {
        _httpClient = httpClient;
    }

    public async Task<List<BggGameSearchResponseDto>> GetBoardGameBySearchStringAsync(string search, CancellationToken cancellationToken)
    {
        try
        {
            var url = $"{_baseUrl}/search?query={search}&type=boardgame";
            var response = await _httpClient.GetAsync(url, cancellationToken);
            response.EnsureSuccessStatusCode();

            var xml = await response.Content.ReadAsStreamAsync(cancellationToken);
            var serializer = new XmlSerializer(typeof(BggGameSearchListResponseDto));
            var gamesResult = (BggGameSearchListResponseDto?)serializer.Deserialize(xml);
            return gamesResult?.Items ?? [];
        }
        catch(IOException)
        {
            return [];
        }
    }

    public async Task<BggGameDetailsItemResponseDto?> GetBoardGameDetailsAsync(int bggId, CancellationToken cancellationToken)
    {
        try
        {
            var url = $"{_baseUrl}/thing?id={bggId}&stats=1";
            var response = await _httpClient.GetAsync(url, cancellationToken);
            response.EnsureSuccessStatusCode();

            var xml = await response.Content.ReadAsStreamAsync(cancellationToken);
            var serializer = new XmlSerializer(typeof(BggGameDetailsListResponseDto));
            var gameDetailsResult = (BggGameDetailsListResponseDto?)serializer.Deserialize(xml);
            return gameDetailsResult?.BoardGames.FirstOrDefault();
        }
        catch (IOException)
        {
            return null;
        }
    }
}
```

Рисунок Г.14 – Слайд №14 (рисунок виконано самостійно)

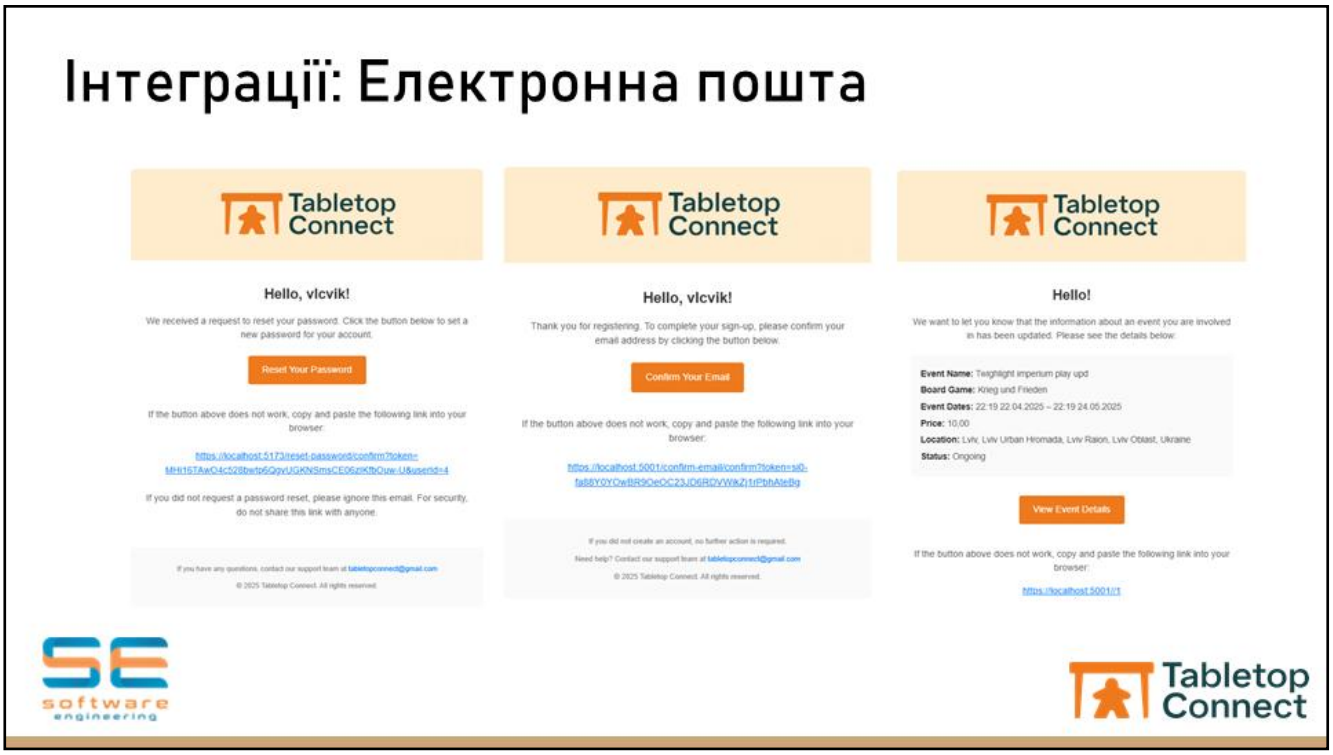


Рисунок Г.15 – Слайд №15 (рисунок виконано самостійно)

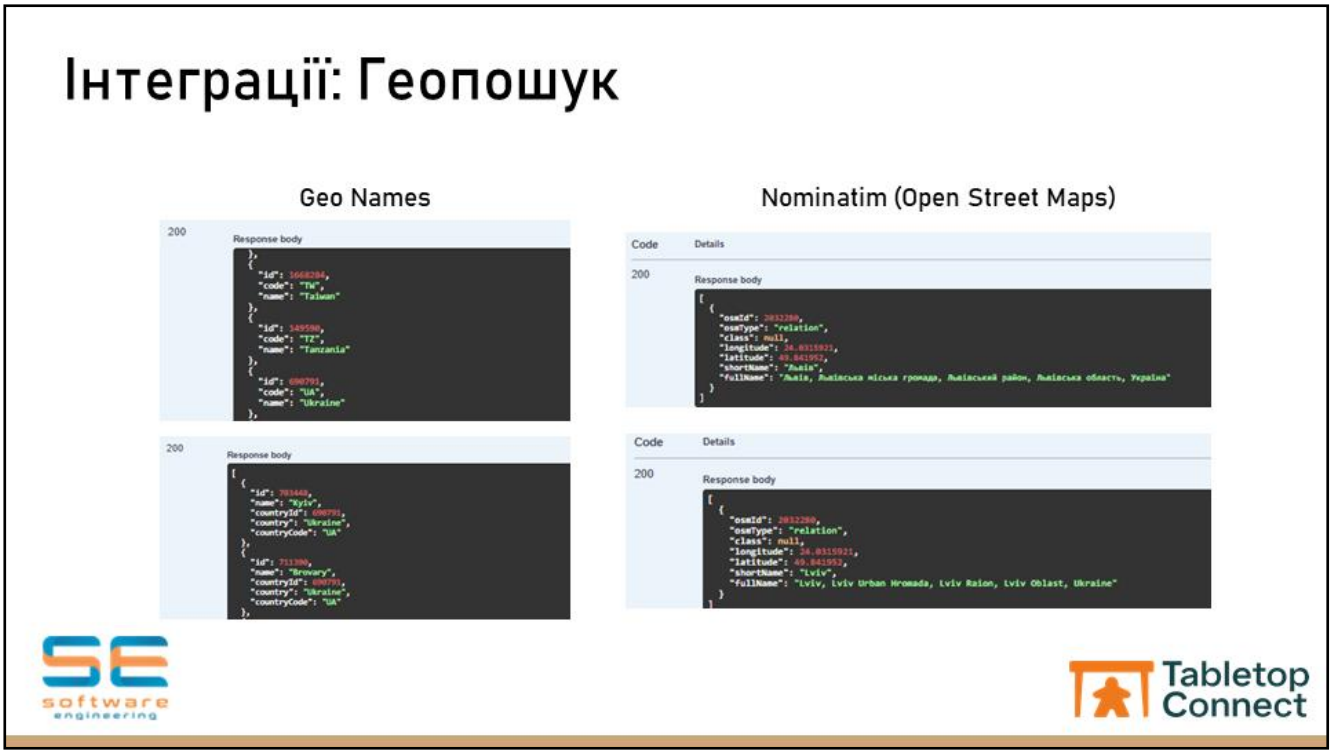


Рисунок Г.16 – Слайд №16 (рисунок виконано самостійно)

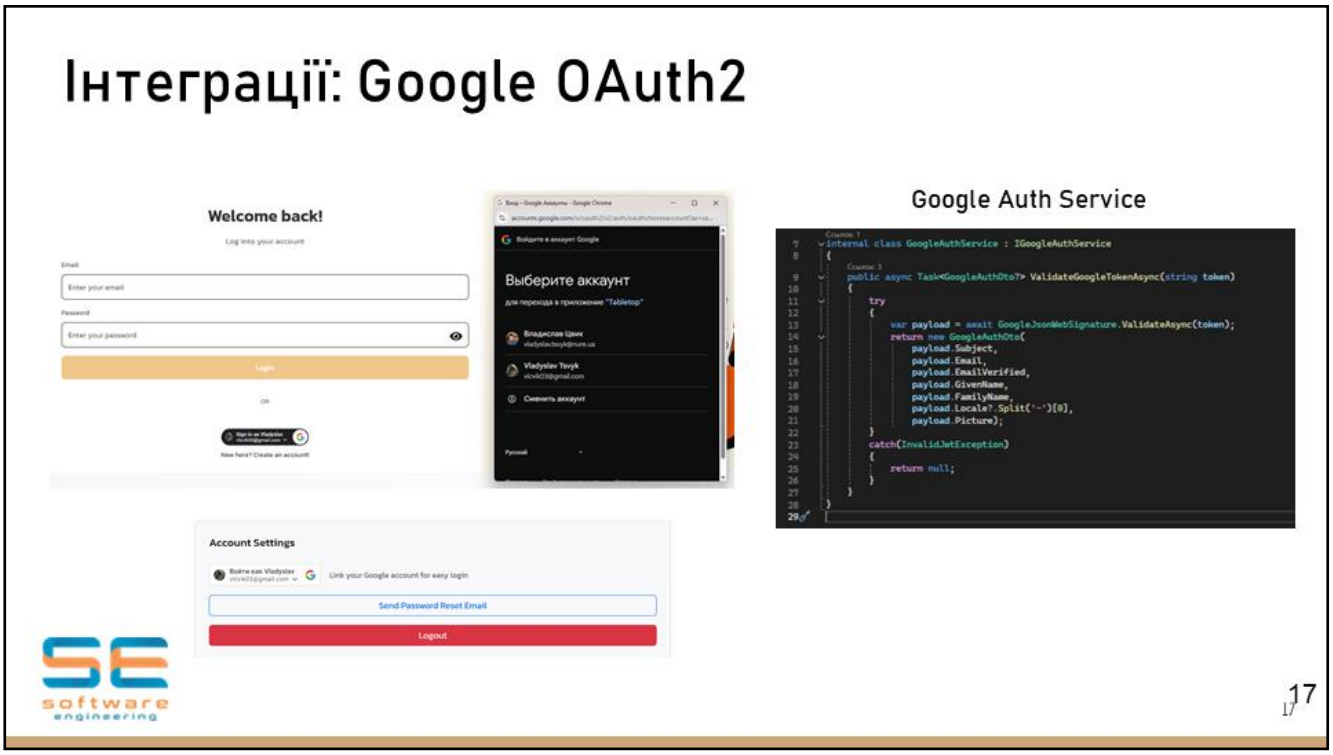


Рисунок Г.17 – Слайд №17 (рисунок виконано самостійно)

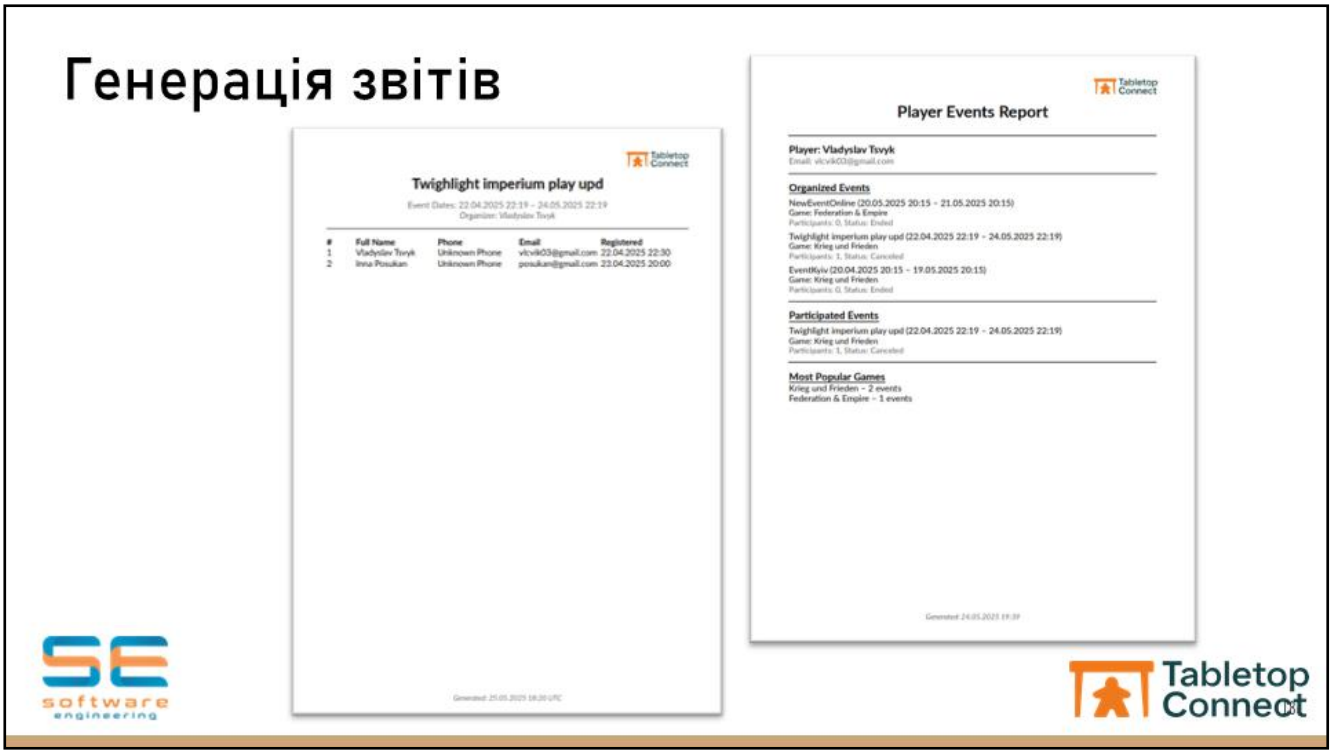
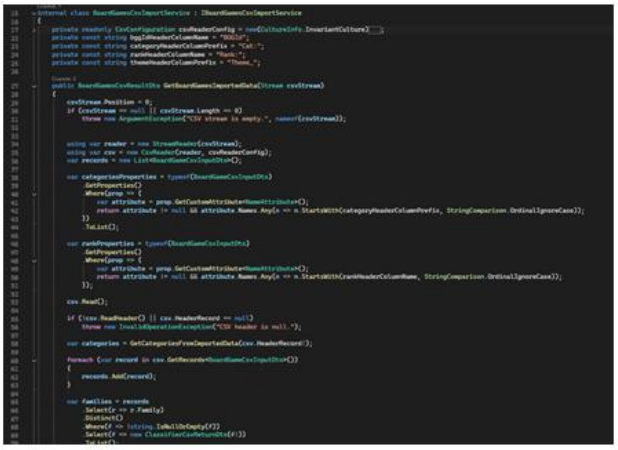


Рисунок Г.18 – Слайд №18 (рисунок виконано самостійно)

Імпорт даних про настільні ігри

BggImportService



- Настільних ігор: > 20,000
- Класифікаторів (категорій, тематики, жанри...): > 5,000
- Відповідностей між іграми та класифікаторами: > 200,000

software engineering

Tabletop Connect

Рисунок Г.19 – Слайд №19 (рисунок виконано самостійно)

Тестування: мануальне

Передумова			
№	Опис випадку	Очікуваний результат	Висновок
1	У базі даних є створена подія, а користувач, який її створив, авторизований	Подію можна скасувати через API	Пройдено
Скасування події			
№	Опис випадку	Очікуваний результат	Висновок
1	Надіслати POST-запит до /api/Events/{id}/cancel автором події	Сервер повертає HTTP-статус 200 (OK), подія позначається як скасована	Пройдено
2	Надіслати POST-запит до /api/Events/{id}/cancel, якщо подія вже скасована	Сервер повертає HTTP-статус 400 (Bad Request) з валідаційною помилкою	Пройдено
3	Надіслати POST-запит до /api/Events/{id}/cancel, де id — неіснуючої події	Сервер повертає HTTP-статус 404 (Not Found) з повідомленням "Подію не знайдено"	Пройдено
4	Надіслати POST-запит до /api/Events/{id}/cancel користувачем, який не є автором	Сервер повертає HTTP-статус 400 (Bad Request) з повідомленням "Немає прав на скасування цієї події"	Пройдено

software engineering

Tabletop Connect

Рисунок Г.20 – Слайд №20 (рисунок виконано самостійно)

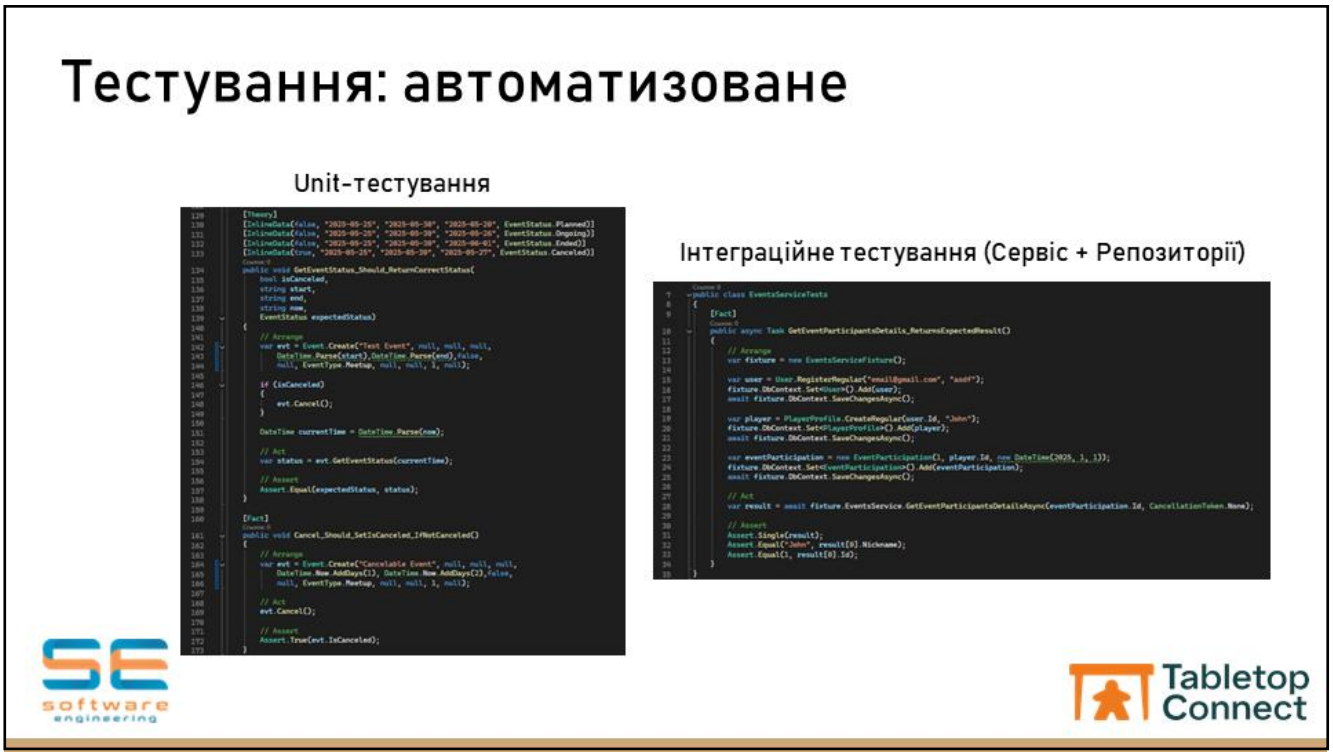


Рисунок Г.21 – Слайд №21 (рисунок виконано самостійно)

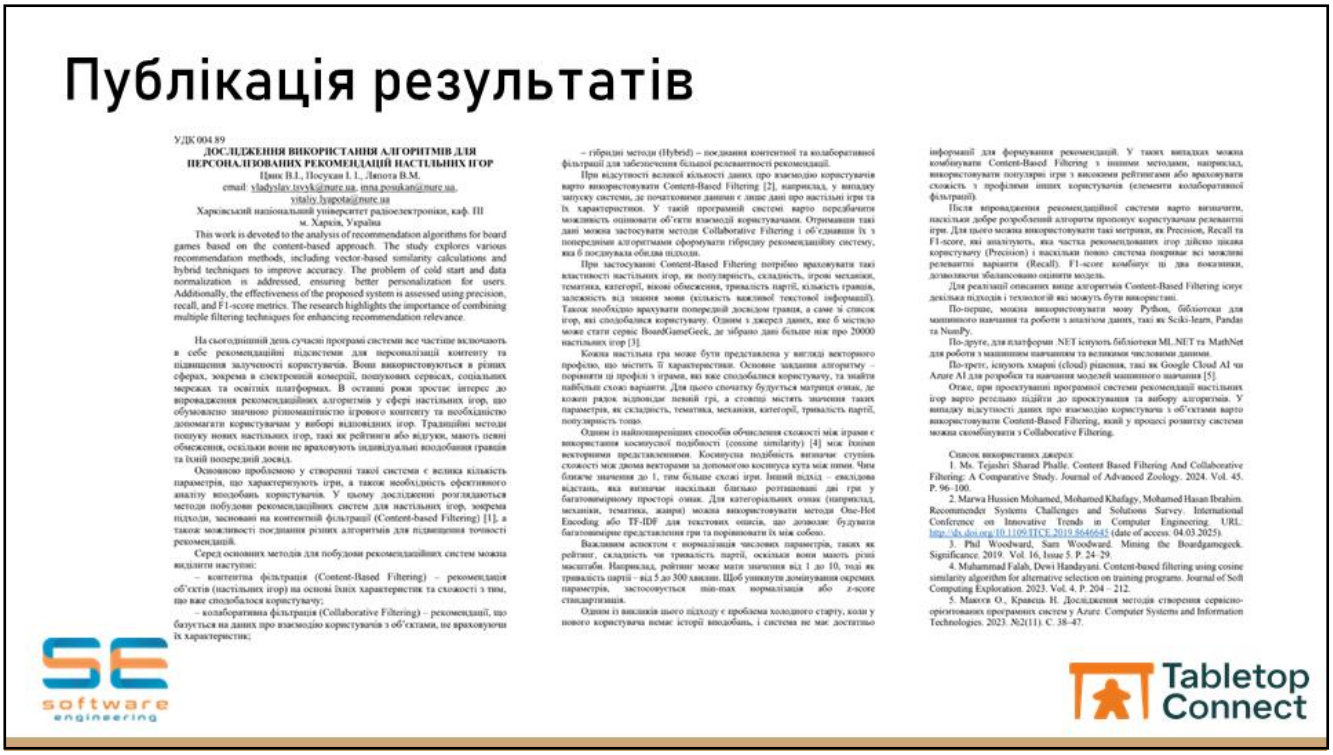


Рисунок Г.22 – Слайд №22 (рисунок виконано самостійно)

Підсумки

- Розроблена система цілком відповідає початковому баченню та поставленим вимогам
- Можливість подальшої монетизації та популяризації
- Створена система може бути повноцінно використана після налаштування інфраструктури
- Подальший розвиток: Mobile (PWA), більше аналітики, просунуті рекомендації

SE software engineering

Tabletop Connect

Рисунок Г.23 – Слайд №23 (рисунок виконано самостійно)

Дякуємо за увагу!

SE software engineering

Tabletop Connect

Рисунок Г.24 – Слайд №24 (рисунок виконано самостійно)

ДОДАТОК Д

Копії тез доповіді на 29-му міжнародному молодіжному форумі

«Радіоелектроніка та молодь У XXI столітті»

УДК 004.89

ДОСЛІДЖЕННЯ ВИКОРИСТАННЯ АЛГОРИТМІВ ДЛЯ ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ НАСТІЛЬНИХ ІГОР

Цвик В.І., Посукан І. І., Ляпота В.М.
email: vladyslav.tsvyk@nure.ua, inna.posukan@nure.ua, vitaliy.lyapota@nure.ua
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна

This work is devoted to the analysis of recommendation algorithms for board games based on the content-based approach. The study explores various recommendation methods, including vector-based similarity calculations and hybrid techniques to improve accuracy. The problem of cold start and data normalization is addressed, ensuring better personalization for users. Additionally, the effectiveness of the proposed system is assessed using precision, recall, and F1-score metrics. The research highlights the importance of combining multiple filtering techniques for enhancing recommendation relevance.

На сьогоднішній день сучасні програмі системи все частіше включають в себе рекомендаційні підсистеми для персоналізації контенту та підвищення залученості користувачів. Вони використовуються в різних сферах, зокрема в електронній комерції, пошукових сервісах, соціальних мережах та освітніх платформах. В останні роки зростає інтерес до впровадження рекомендаційних алгоритмів у сфері настільних ігор, що обумовлено значною різноманітністю ігрового контенту та необхідністю допомагати користувачам у виборі відповідних ігор. Традиційні методи пошуку нових настільних ігор, такі як рейтинги або відгуки, мають певні обмеження, оскільки вони не враховують індивідуальні вподобання гравців та їхній попередній досвід.

Основною проблемою у створенні такої системи є велика кількість параметрів, що характеризують ігри, а також необхідність ефективного аналізу вподобань користувачів. У цьому дослідженні розглядаються методи побудови рекомендаційних систем для настільних ігор, зокрема підходи, засновані на контентній фільтрації (Content-based Filtering) [1], а також можливості поєднання різних алгоритмів для підвищення точності рекомендацій.

Серед основних методів для побудови рекомендаційних систем можна виділити наступні:

– контентна фільтрація (Content-Based Filtering) – рекомендація об’єктів (настільних ігор) на основі їхніх характеристик та схожості з тим, що вже сподобалося користувачу;

– колаборативна фільтрація (Collaborative Filtering) – рекомендації, що базується на даних про взаємодію користувачів з об’єктами, не враховуючи їх характеристик;

– гібридні методи (Hybrid) – поєднання контентної та колаборативної фільтрації для забезпечення більшої релевантності рекомендації.

При відсутності великої кількості даних про взаємодію користувачів варто використовувати Content-Based Filtering [2], наприклад, у випадку запуску системи, де початковими даними є лише дані про настільні ігри та їх характеристики. У такій програмній системі варто передбачити можливість оцінювати об'єкти взаємодії користувачами. Отримавши такі дані можна застосувати методи Collaborative Filtering і об'єднавши їх з попередніми алгоритмами сформувати гібридну рекомендаційну систему, яка б поєднувала обидва підходи.

При застосуванні Content-Based Filtering потрібно враховувати такі властивості настільних ігор, як популярність, складність, ігрові механіки, тематика, категорії, вікові обмеження, тривалість партії, кількість гравців, залежність від знання мови (кількість важливої текстової інформації). Також необхідно врахувати попередній досвідом гравця, а саме зі список ігор, які сподобалися користувачу. Одним з джерел даних, яке б містило може стати сервіс BoardGameGeek, де зібрано дані більше ніж про 20000 настільних ігор [3].

Кожна настільна гра може бути представлена у вигляді векторного профілю, що містить її характеристики. Основне завдання алгоритму – порівняти ці профілі з іграми, які вже сподобалися користувачу, та знайти найбільш схожі варіанти. Для цього спочатку будується матриця ознак, де кожен рядок відповідає певній грі, а стовпці містять значення таких параметрів, як складність, тематика, механіки, категорії, тривалість партії, популярність тощо.

Одним із найпоширеніших способів обчислення схожості між іграми є використання косинусної подібності (cossine similarity) [4] між їхніми векторними представленнями. Косинусна подібність визначає ступінь схожості між двома векторами за допомогою косинуса кута між ними. Чим ближче значення до 1, тим більше схожі ігри. Інший підхід – евклідова відстань, яка визначає наскільки близько розташовані дві гри у багатовимірному просторі ознак. Для категоріальних ознак (наприклад, механіки, тематика, жанри) можна використовувати методи One-Hot Encoding або TF-IDF для текстових описів, що дозволяє будувати багатовимірне представлення гри та порівнювати їх між собою.

Важливим аспектом є нормалізація числових параметрів, таких як рейтинг, складність чи тривалість партії, оскільки вони мають різні масштаби. Наприклад, рейтинг може мати значення від 1 до 10, тоді як тривалість партії – від 5 до 300 хвилин. Щоб уникнути домінування окремих параметрів, застосовується min-max нормалізація або z-score стандартизація.

Одним із викликів цього підходу є проблема холодного старту, коли у нового користувача немає історії вподобань, і система не має достатньо інформації для формування рекомендацій. У таких випадках можна комбінувати Content-Based Filtering з іншими методами, наприклад, використовувати популярні ігри з високими рейтингами або враховувати схожість з профілями інших користувачів (елементи колаборативної фільтрації).

Після впровадження рекомендаційної системи варто визначити, наскільки добре розроблений алгоритм пропонує користувачам релевантні ігри. Для цього можна використовувати такі метрики, як Precision, Recall та F1-score, які

аналізують, яка частка рекомендованих ігор дійсно цікава користувачу (Precision) і наскільки повно система покриває всі можливі релевантні варіанти (Recall). F1-score комбінує ці два показники, дозволяючи збалансовано оцінити модель.

Для реалізації описаних вище алгоритмів Content-Based Filtering існує декілька підходів і технологій які можуть бути використані.

По-перше, можна використовувати мову Python, бібліотеки для машинного навчання та роботи з аналізом даних, такі як Sciki-learn, Pandas та NumPy.

По-друге, для платформи .NET існують бібліотеки ML.NET та MathNet для роботи з машинним навчанням та великими числовими даними.

По-третє, існують хмарні (cloud) рішення, такі як Google Cloud AI чи Azure AI для розробки та навчання моделей машинного навчання [5].

Отже, при проектуванні програмної системи рекомендації настільних ігор варто ретельно підійти до проектування та вибору алгоритмів. У випадку відсутності даних про взаємодію користувача з об’єктами варто використовувати Content-Based Filtering, який у процесі розвитку системи можна скомбінувати з Collaborative Filtering.

Список використаних джерел:

1. Ms. Tejashri Sharad Phalle. Content Based Filtering And Collaborative Filtering: A Comparative Study. Journal of Advanced Zoology. 2024. Vol. 45. P. 96–100.

2. Marwa Hussien Mohamed, Mohamed Khafagy, Mohamed Hasan Ibrahim. Recommender Systems Challenges and Solutions Survey. International Conference on Innovative Trends in Computer Engineering. URL: http://dx.doi.org/10.1109/ITCE.2019.8646645 (date of access: 04.03.2025).

3. Phil Woodward, Sam Woodward. Mining the Boardgamegeek. Significance. 2019. Vol. 16, Issue 5. P. 24–29.

4. Muhammad Falah, Dewi Handayani. Content-based filtering using cosine similarity algorithm for alternative selection on training programs. Journal of Soft Computing Exploration. 2023. Vol. 4. P. 204 – 212.

5. Макєєв О., Кравець Н. Дослідження методів створення сервісно-орієнтованих програмних систем у Azure. Computer Systems and Information Technologies. 2023. №2(11). С. 38–47.