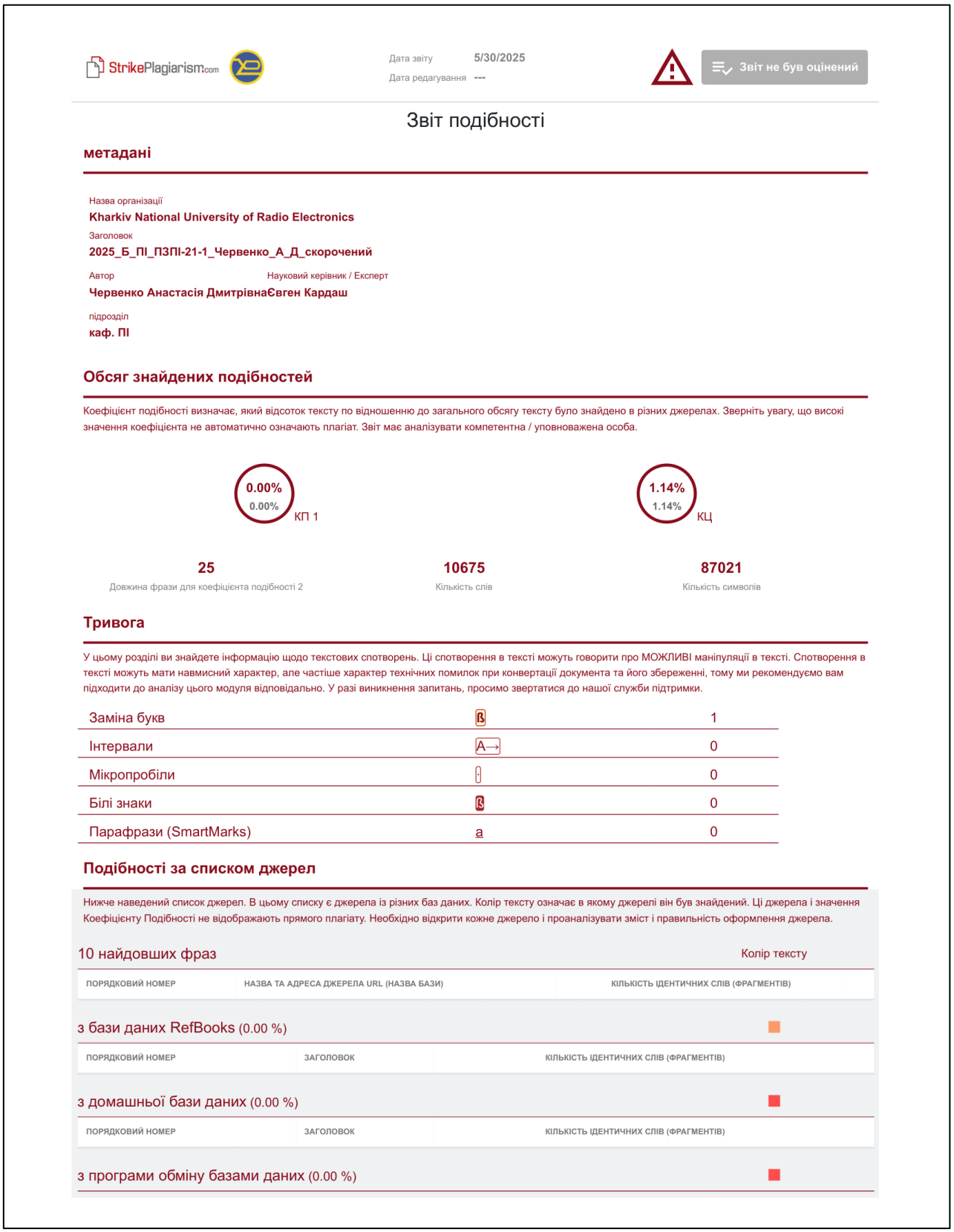
ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



StrikePlagiarism.com

Дата звіту 5/30/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_Б_ПІ_ПЗПІ-21-1_Червенко_А_Д_скорочений
Автор
Червенко Анастасія Дмитрівна
Науковий керівник / Експерт
Євген Кардаш
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.

0.00% 0.00% КП 1

1.14% 1.14% КЦ

25	10675	87021
Довжина фрази для коефіцієнта подібності 2	Кількість слів	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	ß	1
Інтервали	A→	0
Мікропробіли	¦	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	0

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)

з бази даних RefBooks (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з домашньої бази даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з програми обміну базами даних (0.00 %)

Рисунок А.1 – Результат перевірки на унікальність тексту

ДОДАТОК Б

Слайди презентації

Рисунок Б.1 – Слайд 1

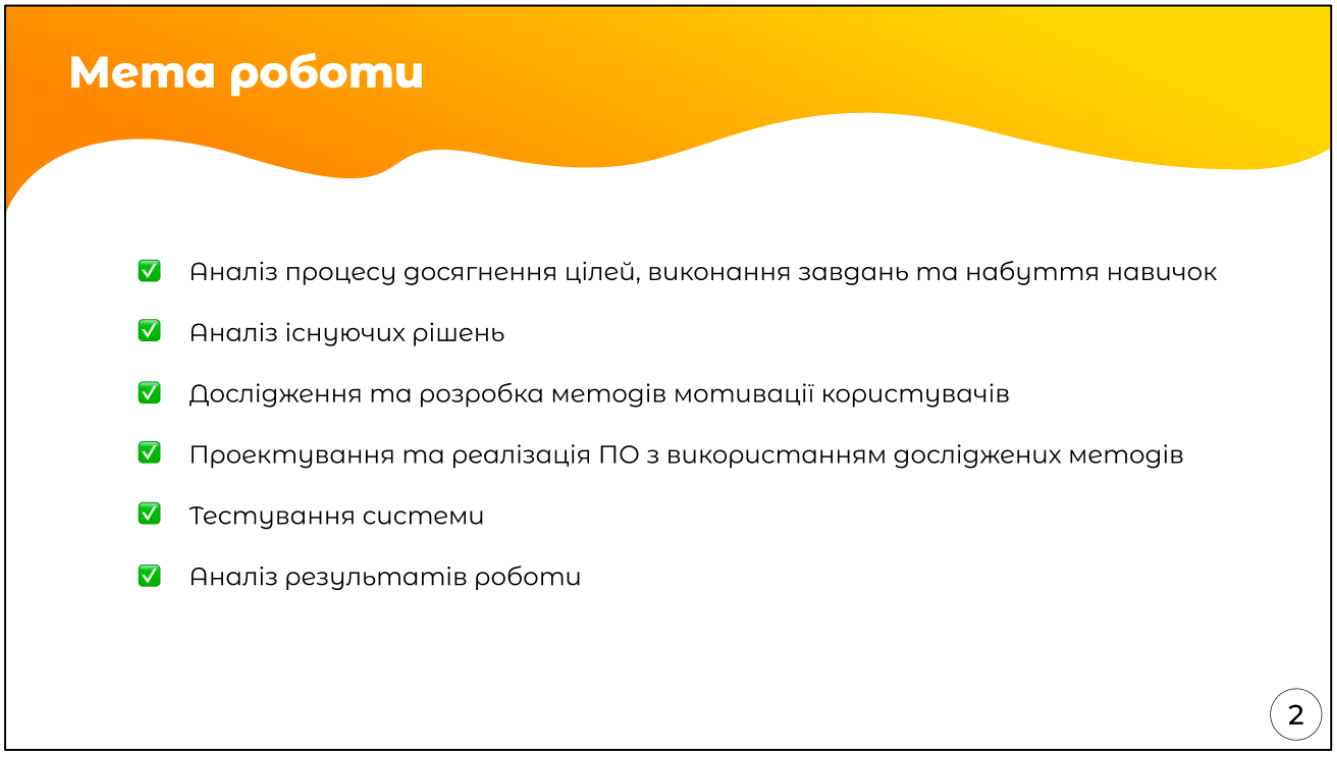


Рисунок Б.2 – Слайд 2

Рисунок Б.3 – Слайд 3

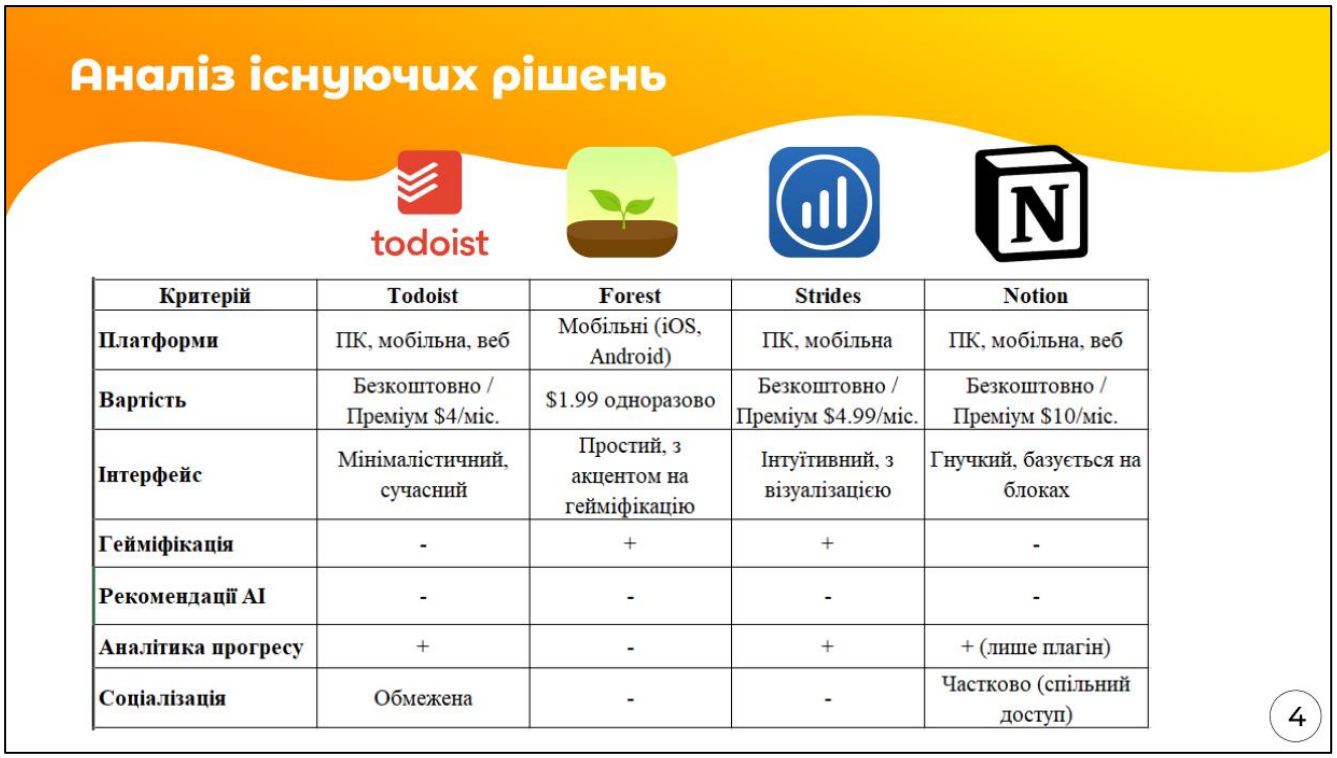


Рисунок Б.4 – Слайд 4

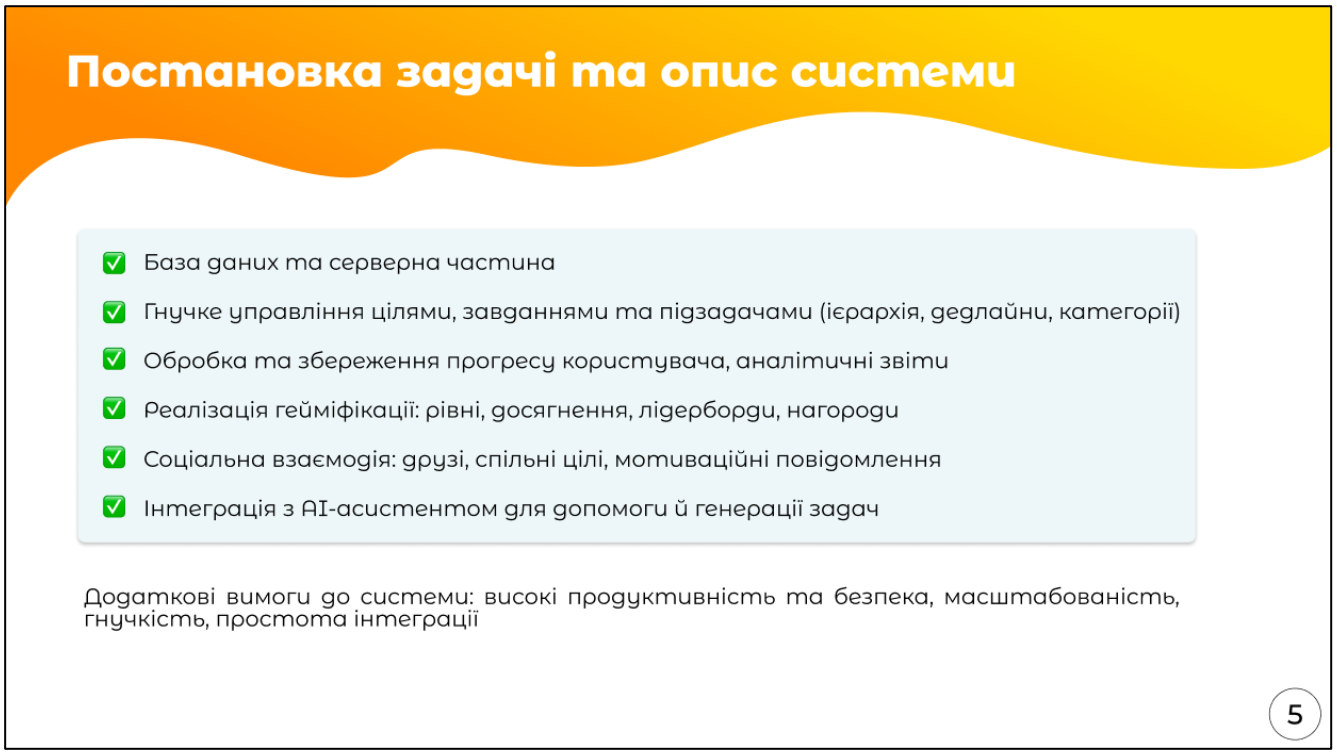


Рисунок Б.5 – Слайд 5

Рисунок Б.6 – Слайд 6

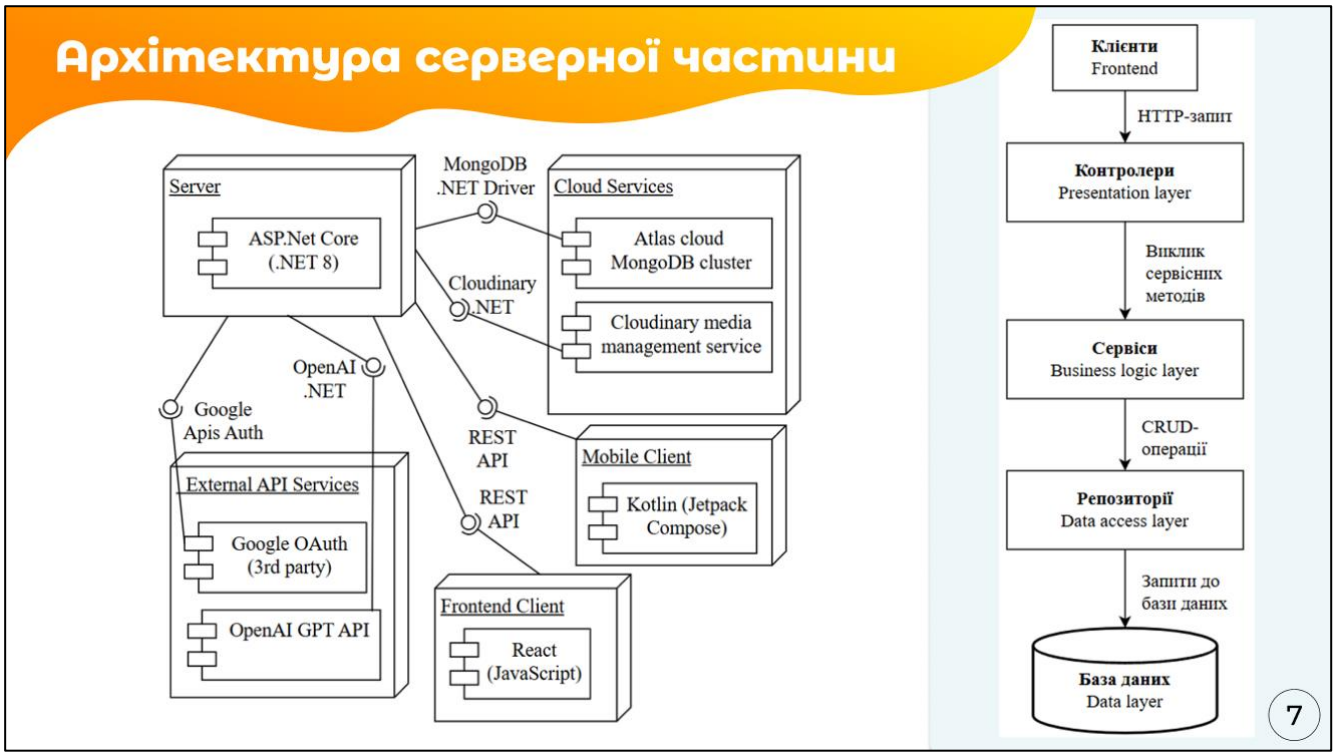


Рисунок Б.7 – Слайд 7

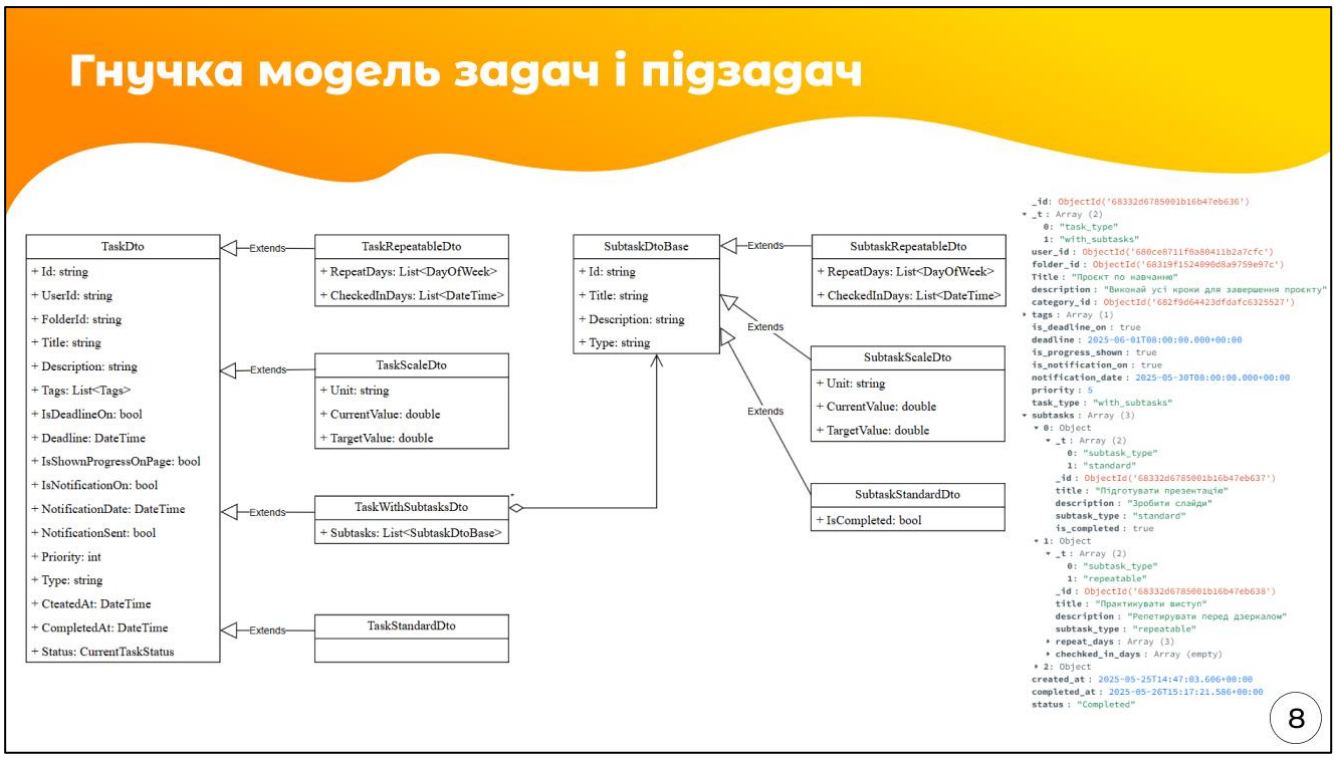


Рисунок Б.8 – Слайд 8

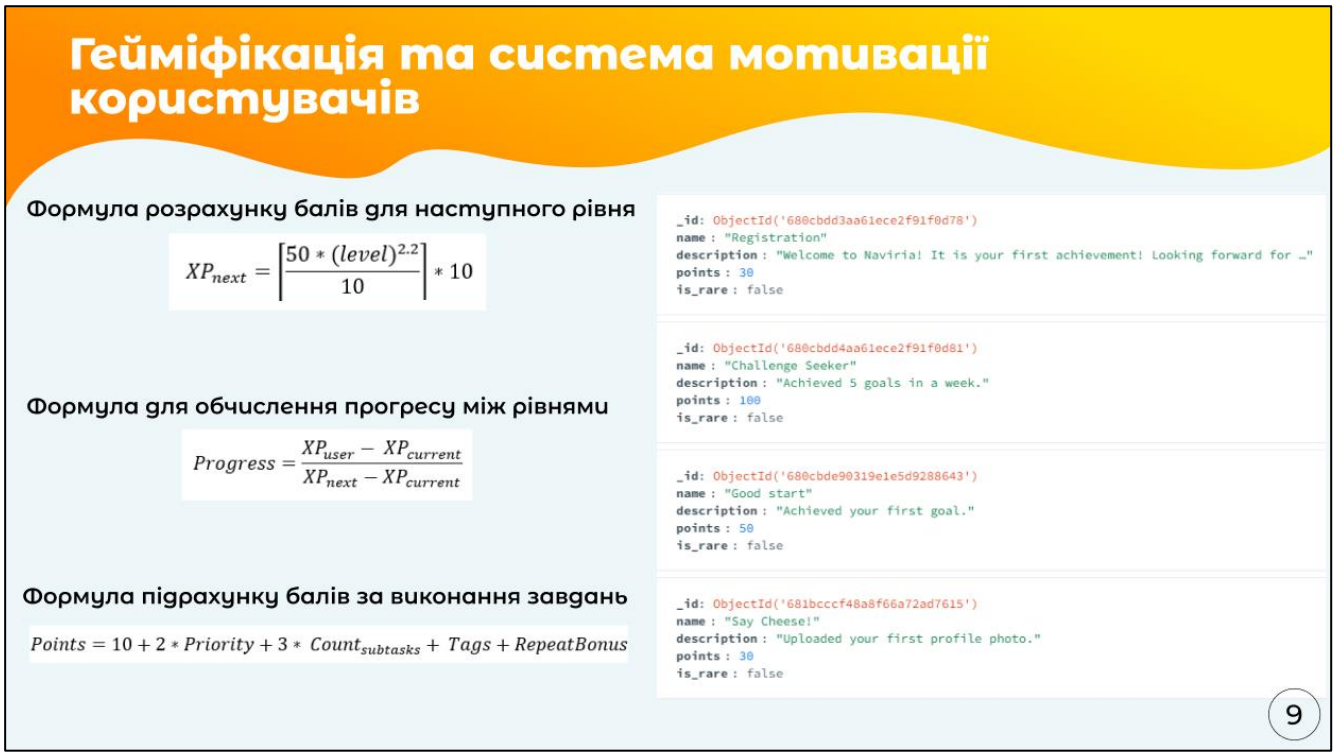


Рисунок Б.9 – Слайд 9

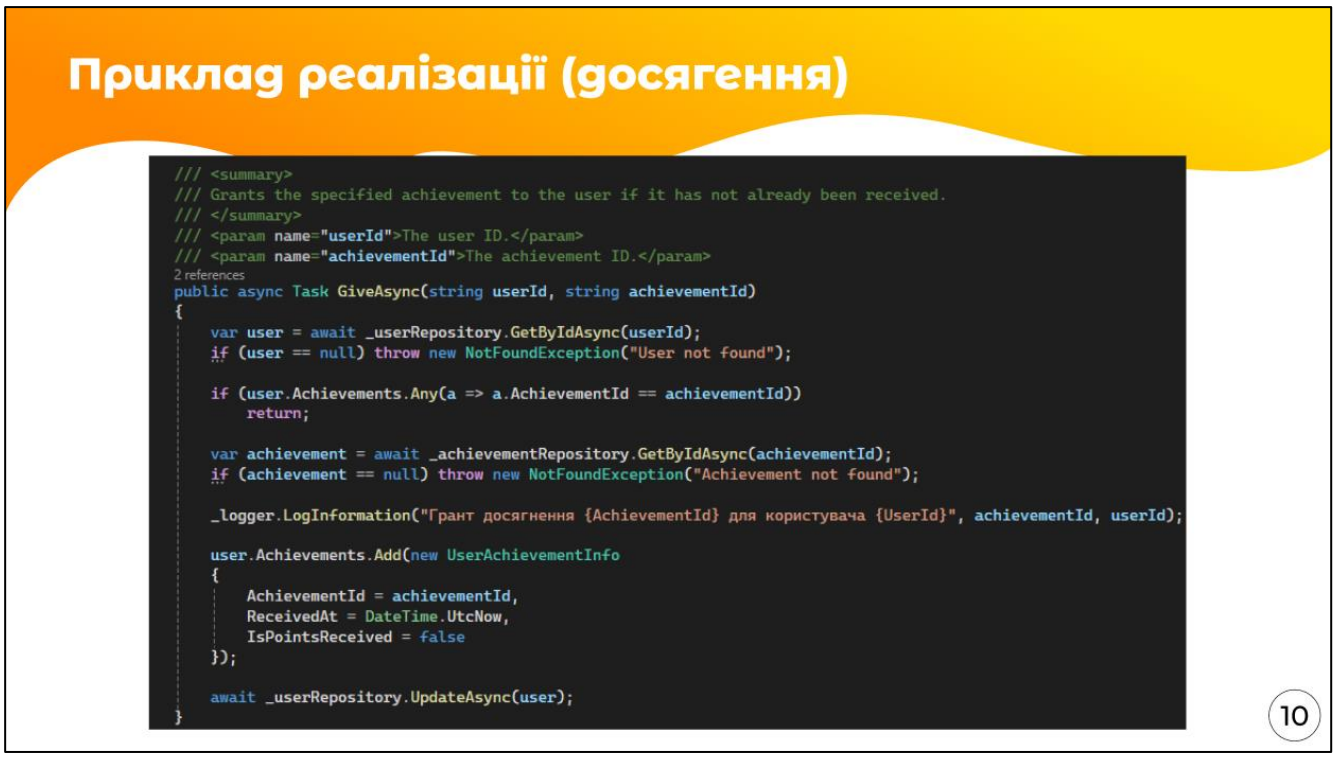


Рисунок Б.10 – Слайд 10

Приклад реалізації (бали за завдання)

```
2 references
public int CalculateTaskPoints(TaskEntity task)
{
    int basePoints = 10;
    int priorityBonus = task.Priority * 2;
    int subtaskBonus = task.Subtasks?.Count * 3 ?? 0;
    int tagsBonus = task.Tags?.Select(t => t.TagName).Distinct().Count() ?? 0;

    int checkedInDaysBonus = 0;
    foreach (var subtask in task.Subtasks)
    {
        if (subtask is SubtaskRepeatable repeatable)
        {
            checkedInDaysBonus += Math.Min(repeatable.CheckedInDays?.Count ?? 0, 10) * 2;
        }
    }

    return basePoints + priorityBonus + subtaskBonus + tagsBonus + checkedInDaysBonus;
}
```

11

Рисунок Б.11 – Слайд 11

Аналітика статистичних даних користувачів

Pie-charts дані виконаних завдань за категоріями та Line-charts дані виконаних завдань за місяцями

- особисті
- серед друзів
- серед усіх користувачів

Змагальна статистика серед користувачів

- топ-10 користувачів системи для дошки лідерів

Загальні статистичні запити

- кількість усіх користувачів
- кількість усіх завдань
- відсоток завершених задач
- кількість днів для користувача з моменту його реєстрації
- отримати кількість днів з моменту зародження проекту

12

Рисунок Б.12 – Слайд 12

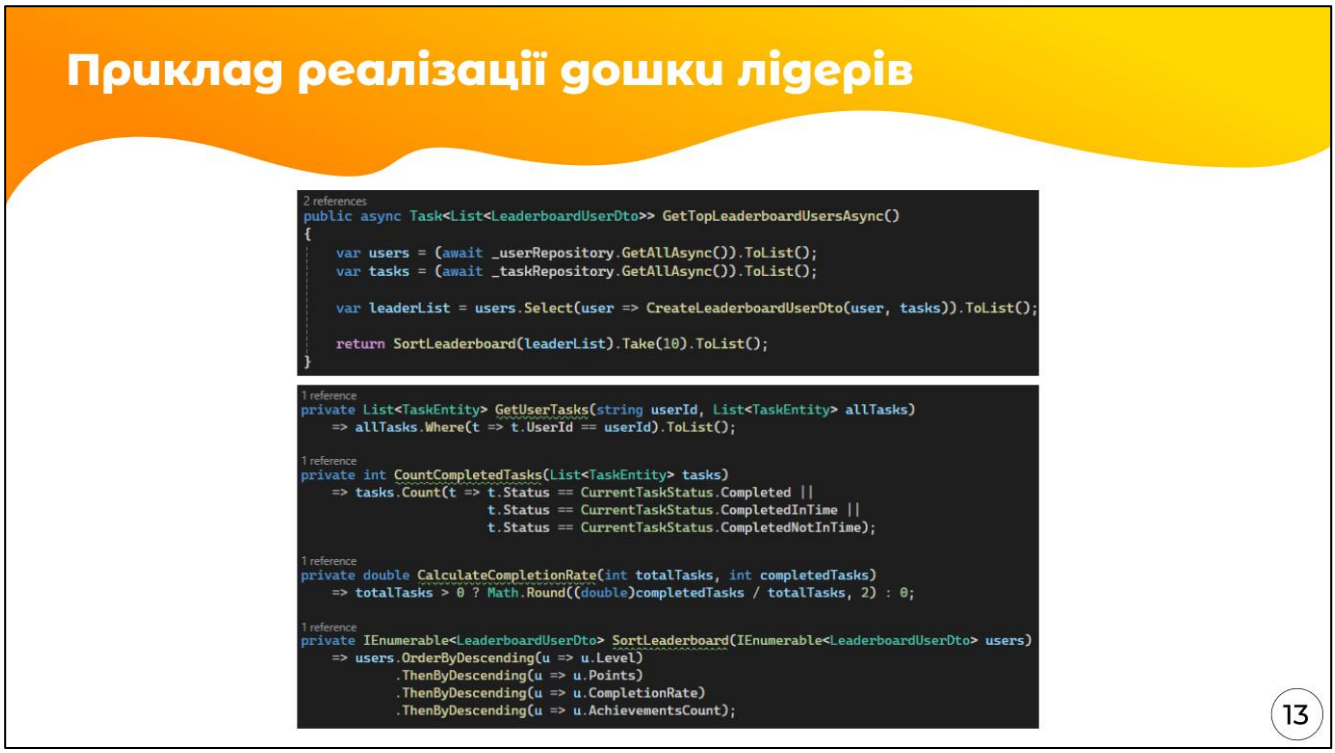


Рисунок Б.13 – Слайд 13

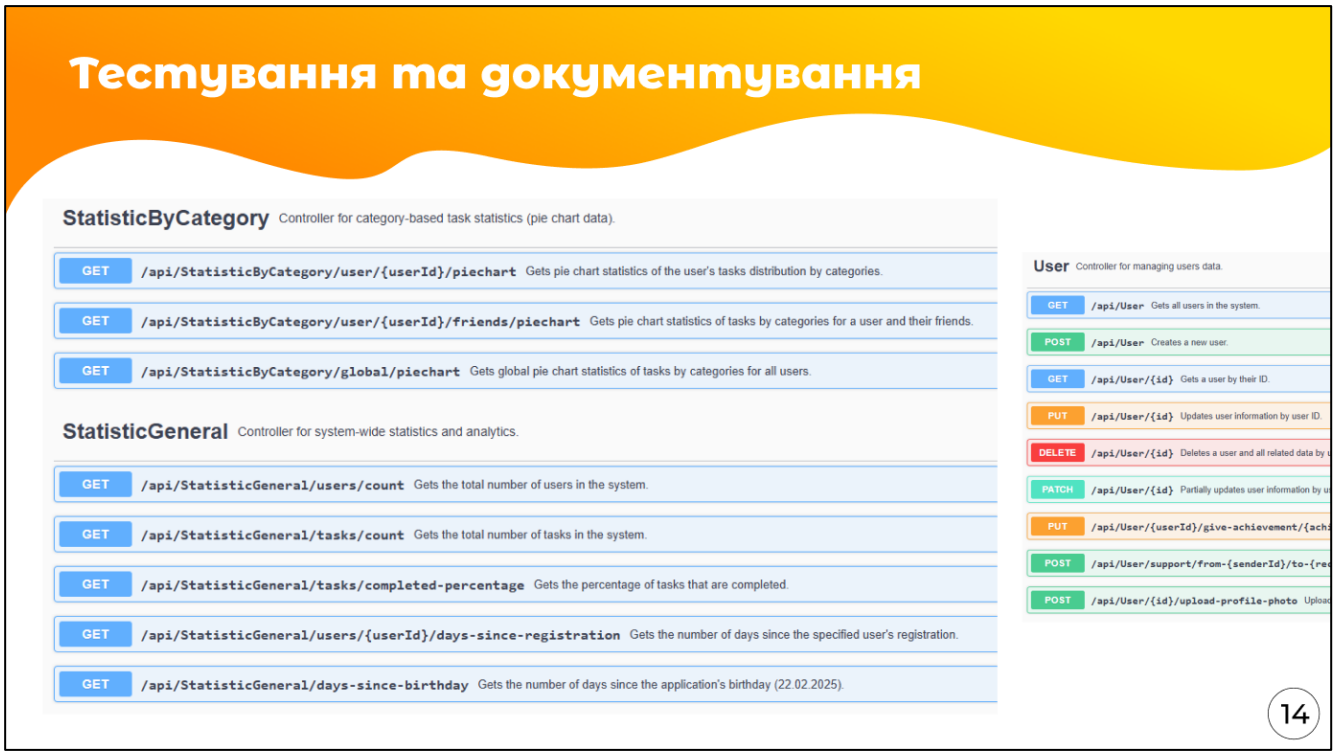


Рисунок Б.14 – Слайд 14

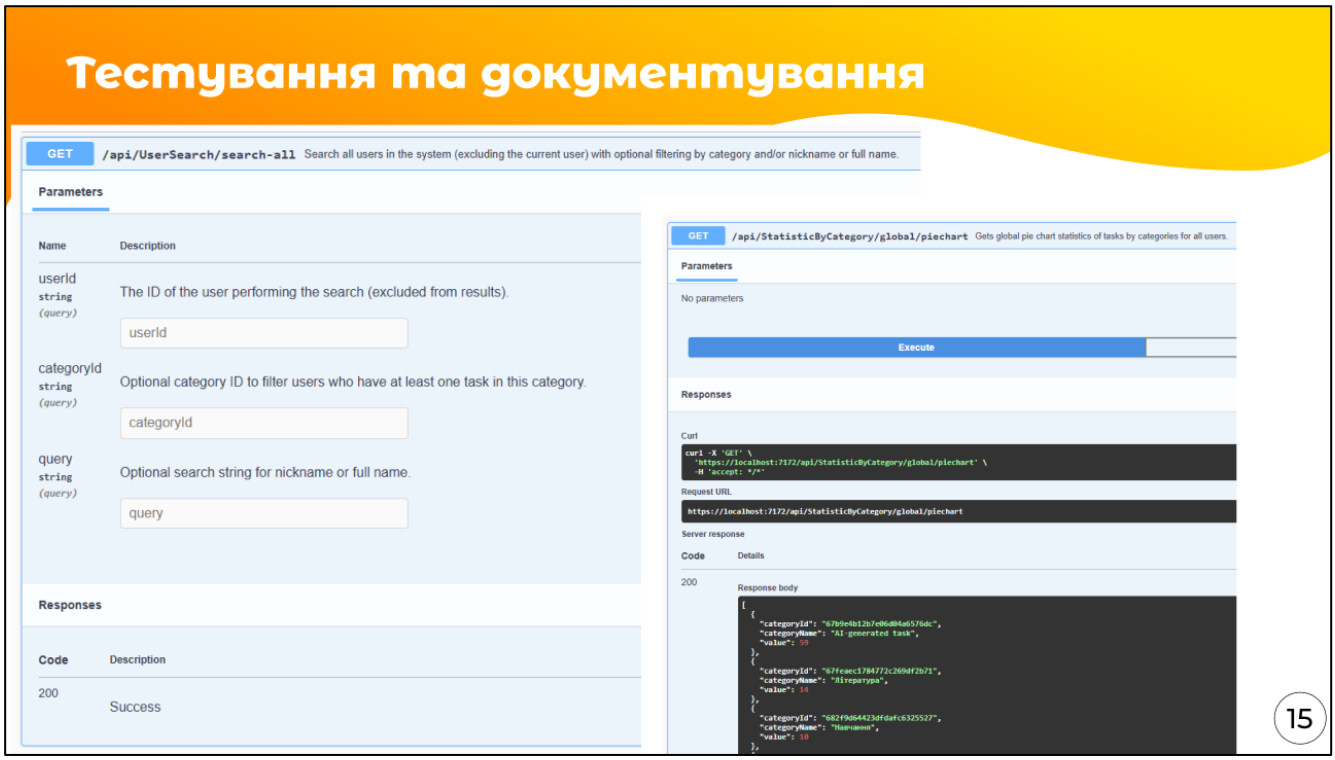


Рисунок Б.15 – Слайд 15

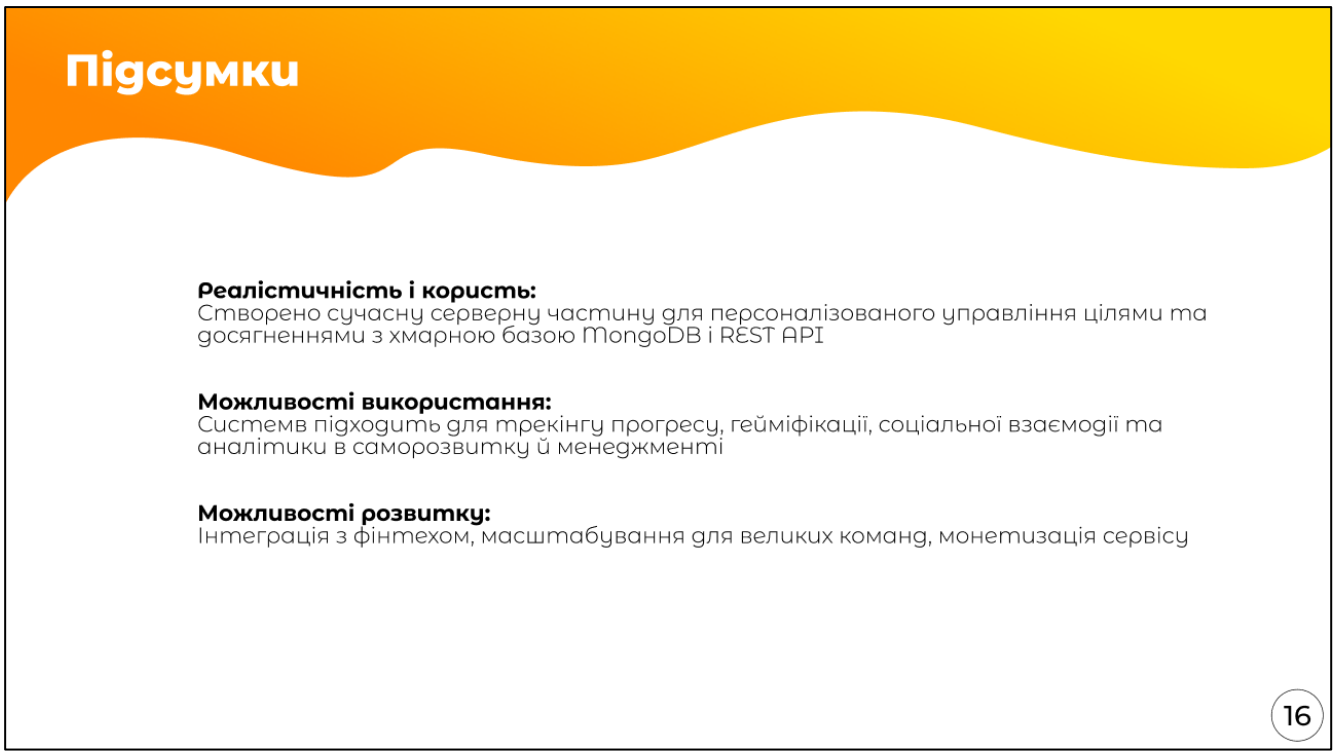


Рисунок Б.16 – Слайд 16

ДОДАТОК В

Таблиці

	Todoist	Habitica	Forest	Strides	Notion	Trello	Naviria
Платформи	ПК, мобільна, веб	ПК, мобільна	Мобільні (iOS, Android)	ПК, мобільна	ПК, мобільна, веб	ПК, мобільна, веб	ПК, мобільна, веб
Смарт-годинники	+	-	+	-	-	-	-
Вартість	Безкоштовно / Преміум $4/міс.	Безкоштовно / Преміум $5/міс.	$1.99 одноразово	Безкоштовно / Преміум $4.99/міс.	Безкоштовно / Преміум $10/міс.	Безкоштовно / Преміум $5/міс.	Безкоштовно
Інтерфейс	Мінімалістичний, сучасний	Стилізований під RPG	Простий, з акцентом на гейміфікацію	Інтуїтивний, із візуалізацією	Гнучкий, базується на блоках	Простий, із картковим підходом	Сучасний, з акцентом на аналітику та AI
Геймфікація	-	+	+	-	-	-	+
Рекомендації AI	-	-	-	-	-	-	+
Аналітика прогресу	+	-	-	+	- (лише плагіни)	-	+
Соціалізація	Обмежена	Присутня (групові завдання)	-	-	Частково (спільний доступ)	Присутня (командна робота)	+

Рисунок В.1 – Порівняння функціональних можливостей конкурентів з програмною системою «Naviria»

Таблиця В.2 – Повний список та опис усіх реалізованих ендпоінтів

Controller	HTTP Method	Endpoint	Опис
Achievements	GET	/api/Achievements	Отримати список усіх досягнень
	GET	/api/Achievements/{id}	Отримати досягнення за ідентифікатором
	POST	/api/Achievements	Створити нове досягнення
	PUT	/api/Achievements/{id}	Оновити дані про досягнення за ідентифікатором
	DELETE	/api/Achievements/{id}	Видалити досягнення та прибрати його у всіх користувачів, які його мали
	GET	/api/Achievements/user /{userId}	Отримати всі досягнення конкретного користувача
	PUT	/api/Achievements/{userId} /award-achievement-points/ {achievementId}	Нарахувати користувачу бали за конкретне досягнення
AssistantChat	GET	/api/AssistantChat/user/ {userId}	Отримати всі повідомлення персонального чату користувача з асистентом
	POST	/api/AssistantChat/ask	Надіслати повідомлення асистенту (ChatGPT) від імені користувача, отримати відповідь
Auth	POST	/api/Auth/login	Авторизація користувача за email та паролем, отримання JWT-токена.
	POST	/api/Auth/google-login	Авторизація користувача через Google OAuth, отримання JWT-токена
Backup	POST	/api/Backup/export-db-data	Створити резервну копію всієї бази даних, отримати файл архіву (.gz)
	GET	/api/Backup/list	Отримати список доступних резервних копій
	POST	/api/Backup/import-db-data	Відновити всю базу даних із вказаного резервного архіву
	POST	/api/Backup/import-collection	Відновити окрему колекцію з останньої резервної копії (за назвою колекції)
CloudUpload	POST	/api/CloudUploading/ upload-image	Завантажити зображення у хмарне сховище
Category	GET	/api/Category	Отримати список усіх категорій.
	GET	/api/Category/{id}	Отримати категорію за її ідентифікатором.
	POST	/api/Category	Створити нову категорію.
	PUT	/api/Category/{id}	Оновити дані про категорію за ідентифікатором.
	DELETE	/api/Category/{id}	Видалити категорію та всі її пов’язані задачі за ідентифікатором

Продовження таблиці В.2

Folder	GET	/api/Folder/user/{id}	Отримати всі папки для вказаного користувача за ідентифікатором
	GET	/api/Folder/{id}	Отримати папку за її ідентифікатором
	POST	/api/Folder	Створити нову папку
	PUT	/api/Folder/{id}	Оновити дані про папку за ідентифікатором
	DELETE	/api/Folder/{id}	Видалити папку та всі задачі, які містяться в ній, за ідентифікатором
FriendRequest	GET	/api/FriendRequest	Отримати список усіх запитів у друзі
	GET	/api/FriendRequest/{id}	Отримати запит у друзі за його ідентифікатором
	POST	/api/FriendRequest	Створити новий запит у друзі
	PUT	/api/FriendRequest/{id}	Оновити дані запиту у друзі за ідентифікатором
	DELETE	/api/FriendRequest/{id}	Видалити запит у друзі за ідентифікатором
	GET	/api/FriendRequest /incoming/{userId}	Отримати всі вхідні запити у друзі для вказаного користувача
Friends	GET	/api/Friends/{id}	Отримати список друзів для заданого користувача (за його ID)
	DELETE	/api/Friends/{fromUserId} /to/{friendId}	Видалити користувача з друзів (два ID: хто видаляє і кого видаляють)
	GET	/api/Friends/{id} /potential-friends	Отримати перелік потенційних друзів (тих, хто ще не в друзях у користувача)
	GET	/api/Friends/{userId} /search-friends?query=...	Пошук користувачів за нікнеймом (не включає себе та поточних друзів)
	GET	/api/Friends/shared-interests /{userId1}/{userId2}	Знайти спільні інтереси (категорії задач і теги) між двома користувачами
Leaderboard	GET	/api/Leaderboard/top	Отримати топ-10 користувачів у дошці лідерів разом зі статистикою по кожному з них
Notification	GET	/api/Notification	Отримати всі нотифікації в системі.
	GET	/api/Notification/{id}	Отримати нотифікацію за її ідентифікатором
	POST	/api/Notification	Створити нову нотифікацію
	PUT	/api/Notification/user /{userId}/mark-all-read	Позначити всі нотифікації користувача як прочитані
	DELETE	/api/Notification/{id}	Видалити нотифікацію за ідентифікатором
	GET	/api/Notification/user/ {userId}	Отримати всі нотифікації для конкретного користувача

Продовження таблиці В.2

Quote	GET	/api/Quote	Отримати список усіх цитат
	GET	/api/Quote/{id}	Отримати цитату за її ID
	POST	/api/Quote	Створити нову цитату
	PUT	/api/Quote/{id}	Оновити дані про цитату за ID
	DELETE	/api/Quote/{id}	Видалити цитату за ID
StatisticBy Category	GET	/api/StatisticByCategory /user/{userId}/piechart	Отримати дані для кругової діаграми (pie chart) по категоріях задач конкретного користувача
	GET	/api/StatisticByCategory /user/{userId}/friends /piechart	Отримати дані для pie chart по категоріях задач користувача і його друзів
	GET	/api/StatisticByCategory /global/piechart	Отримати глобальні статистичні дані для pie chart по категоріях задач усіх користувачів
Statistic General	GET	/api/StatisticGeneral/users /count	Отримати загальну кількість користувачів у системі
	GET	/api/StatisticGeneral/tasks /count	Отримати загальну кількість задач у системі
	GET	/api/StatisticGeneral/tasks /completed-percentage	Отримати відсоток завершених задач (від 0 до 100).
	GET	/api/StatisticGeneral/users /{userId}/days-since-registration	Отримати кількість днів із моменту реєстрації користувача
	GET	/api/StatisticGeneral/days-since-birthday	Отримати кількість днів із дня народження застосунку (22.02.2025)
StatisticsTask ByDate	GET	/api/StatisticsTaskByDate /user/{userId}/completed /monthly	Отримати статистику завершених задач по місяцях (лінійний графік) для конкретного користувача
	GET	/api/StatisticsTaskByDate/user/{userId}/friends /completed/monthly	Отримати статистику завершених задач по місяцях для користувача і його друзів (лінійний графік)
	GET	/api/StatisticsTaskByDate /global/completed/monthly	Отримати глобальну статистику завершених задач по місяцях для всіх користувачів (лінійний графік)
StatisticTask CheckIn	GET	/api/StatisticTaskCheckIn /checkedin/total/{userId}	Отримати загальну кількість днів чек-іну для всіх повторюваних підзадач користувача
	GET	/api/StatisticTaskCheckIn /checkedin/{userId} /{subtaskId}	Отримати кількість днів чек-іну для конкретної повторюваної підзадачі користувача
Subtask	POST	/api/tasks/{taskId}/subtasks	Додати нову підзадачу до задачі за її ID
	PUT	/api/tasks/{taskId} /subtasks/{subtaskId}	Оновити підзадачу за її ID у задачі
	DELETE	/api/tasks/{taskId} /subtasks/{subtaskId}	Видалити підзадачу за її ID у задачі
	POST	/api/tasks/{taskId}/subtasks /{subtaskId}/checkin	Позначити повторювану підзадачу як виконану (чек-ін) за певну дату.

Кінець таблиці В.2

Task	GET	/api/Task/user/{userId}	Отримати всі задачі для вказаного користувача
	GET	/api/Task/{id}	Отримати задачу за її ідентифікатором
	POST	/api/Task	Створити нову задачу
	PUT	/api/Task/{id}	Оновити дані задачі за ідентифікатором
	DELETE	/api/Task/{id}	Видалити задачу за ідентифікатором
	GET	/api/Task/grouped/user /{userId}	Отримати всі задачі користувача, згруповані по папках
	POST	/api/Task/{taskId}/checkin	Позначити повторювану задачу як виконану (чек-ін) за певну дату
User	GET	/api/User	Отримати всіх користувачів системи
	GET	/api/User/{id}	Отримати користувача за його ідентифікатором
	POST	/api/User	Створити нового користувача (повертає JWT токен)
	PUT	/api/User/{id}	Оновити дані користувача за ідентифікатором
	PATCH	/api/User/{id}	Часткове оновлення даних користувача (оновлюються лише ненульові поля)
	DELETE	/api/User/{id}	Видалити користувача та всі пов’язані з ним дані за ідентифікатором.
	PUT	/api/User/{userId} /give-achievement /{achievementId}	Видати користувачу досягнення за ідентифікатором
	POST	/api/User/support/from-{senderId}/to-{receiverId}	Надіслати повідомлення підтримки від одного користувача іншому
	POST	/api/User/{id}/upload-profile-photo	Завантажити нове фото профілю для користувача
UserSearch	GET	/api/UserSearch/search-all	Пошук усіх користувачів у системі (крім себе), з опціональною фільтрацією за категорією й/або ім’ям/нікнеймом
	GET	/api/UserSearch/search-friends	Пошук серед друзів користувача, з опціональною фільтрацією за категорією й/або ім’ям/нікнеймом
	GET	/api/UserSearch /search-incoming-requests	Пошук серед користувачів, які надіслали запит у друзі, з опціональною фільтрацією за категорією й/або ім’ям/нікнеймом

ДОДАТОК Г

Діаграми

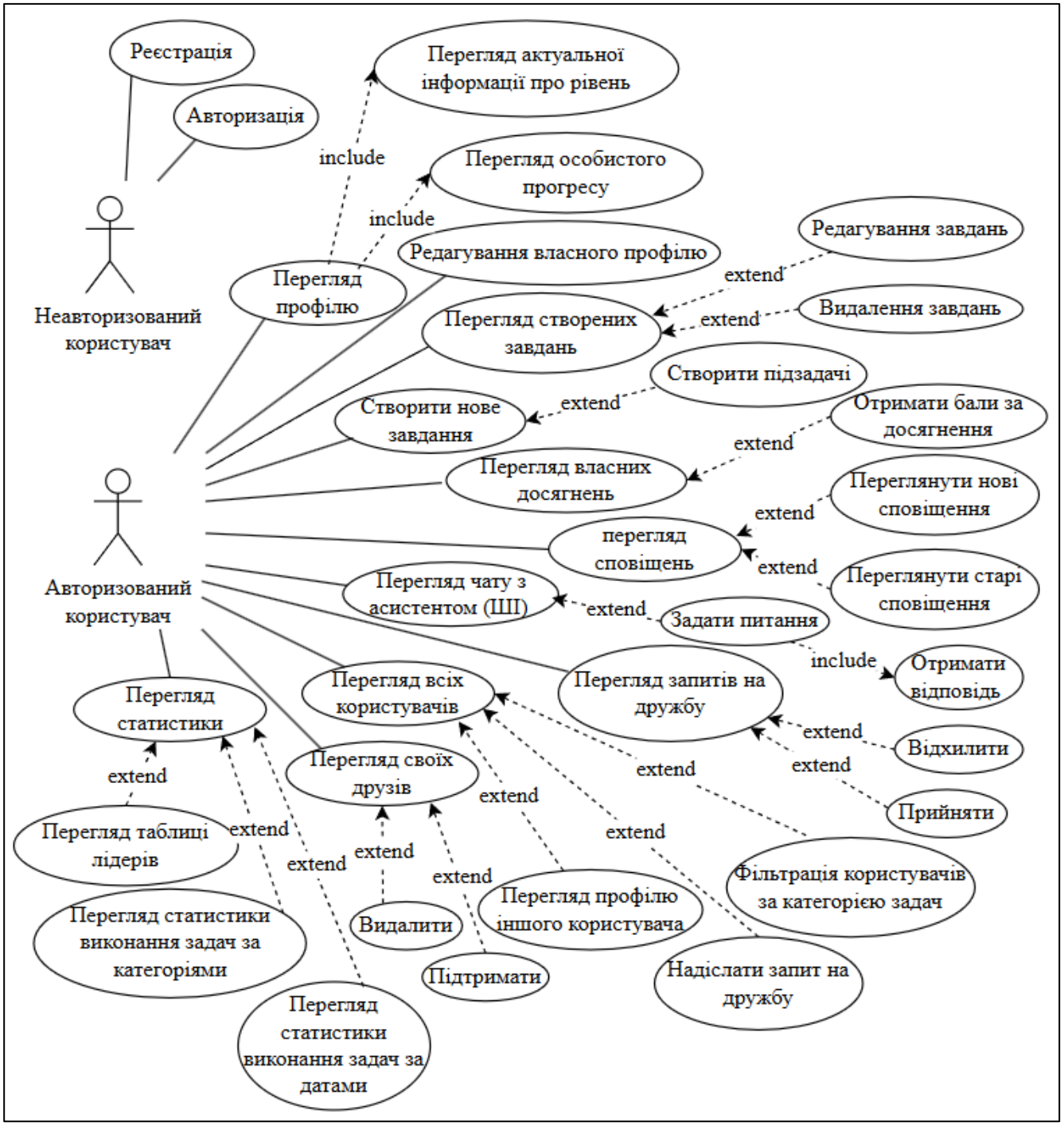


Рисунок Г.1 – Діаграма прецедентів з акторами авторизованого та неавторизованого користувача програмної системи «Naviria»

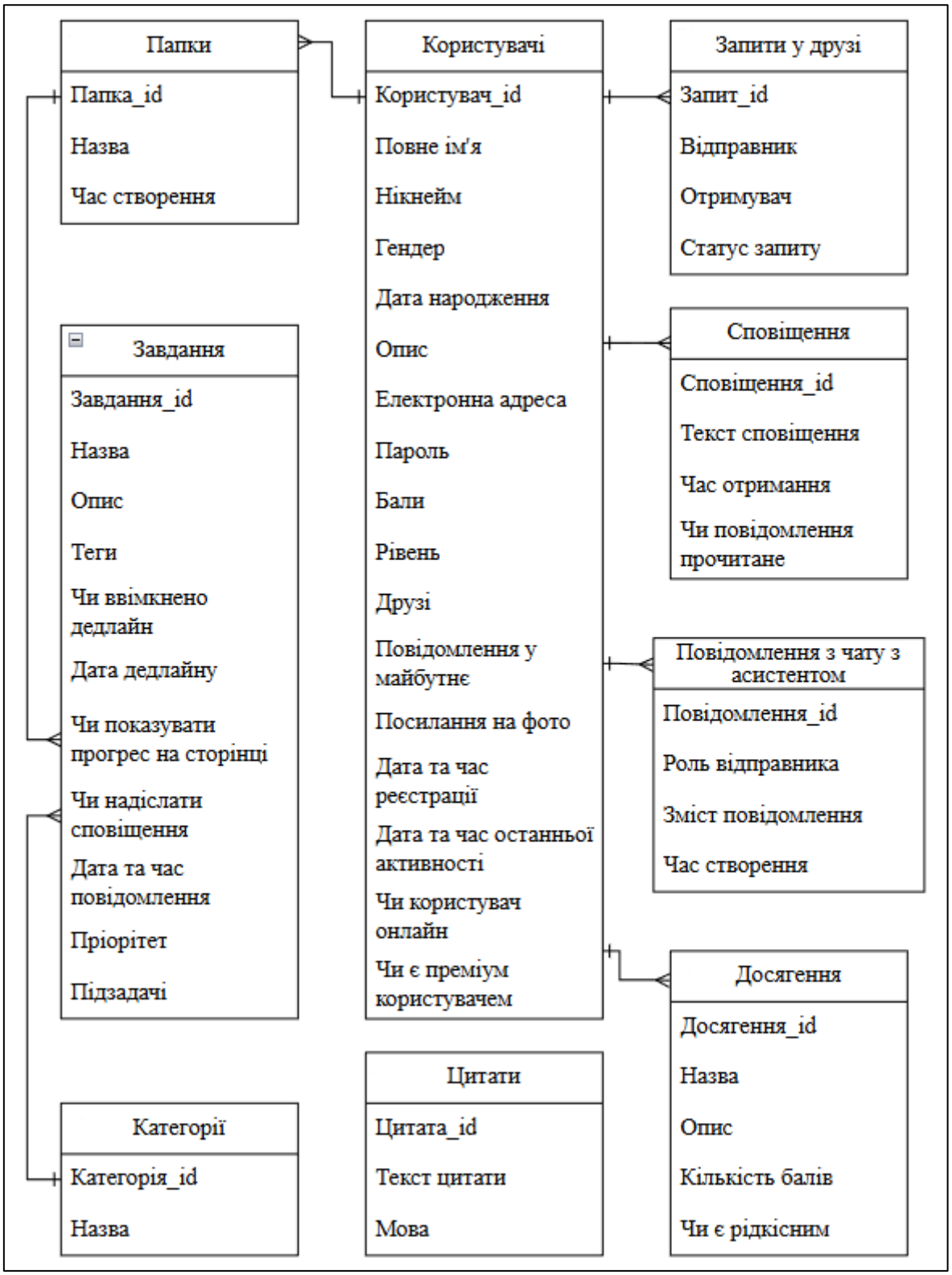


Рисунок Г.2 – ER-діаграма програмної системи «Naviria»

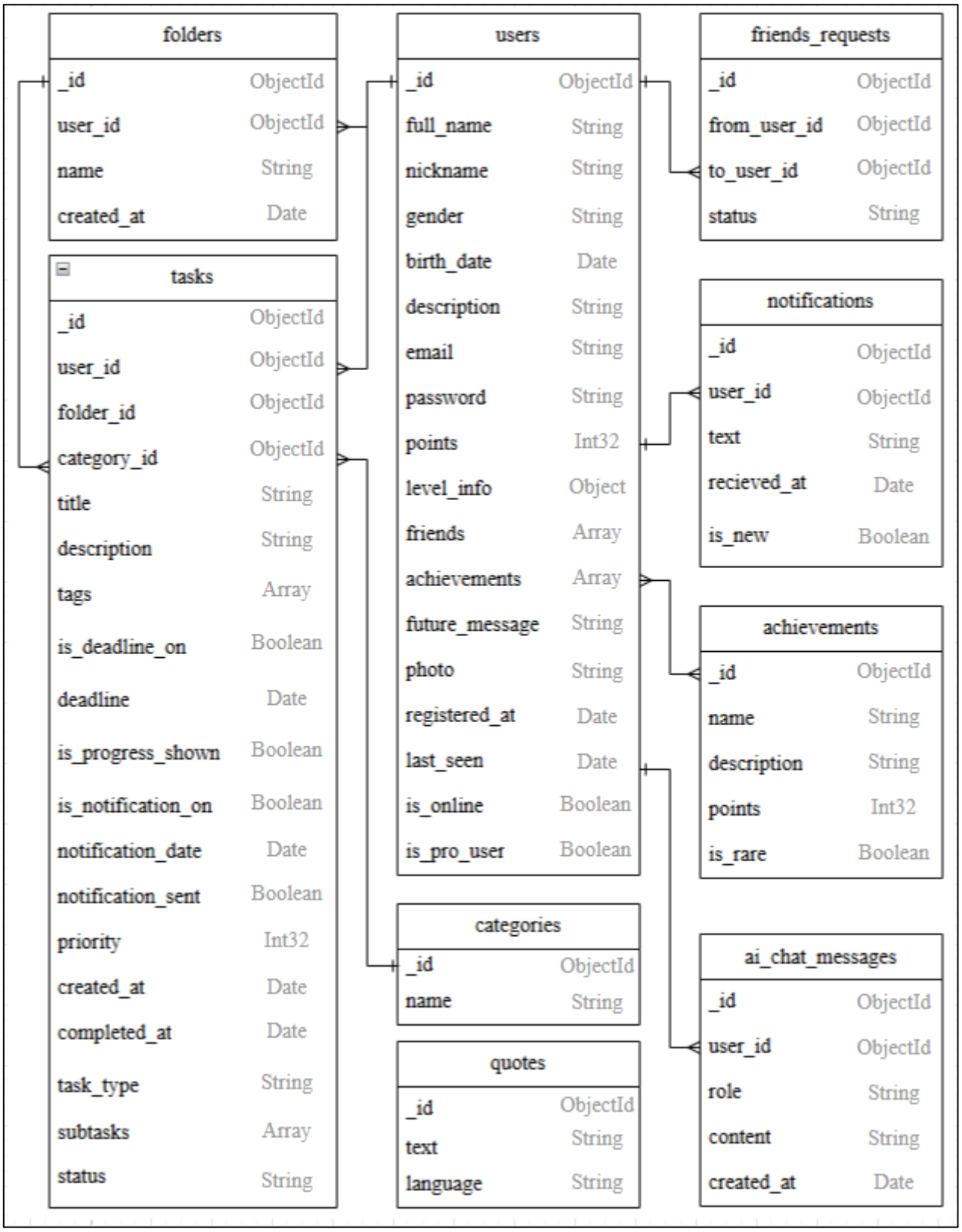


Рисунок Г.3 – Схема бази даних програмної системи «Naviria»

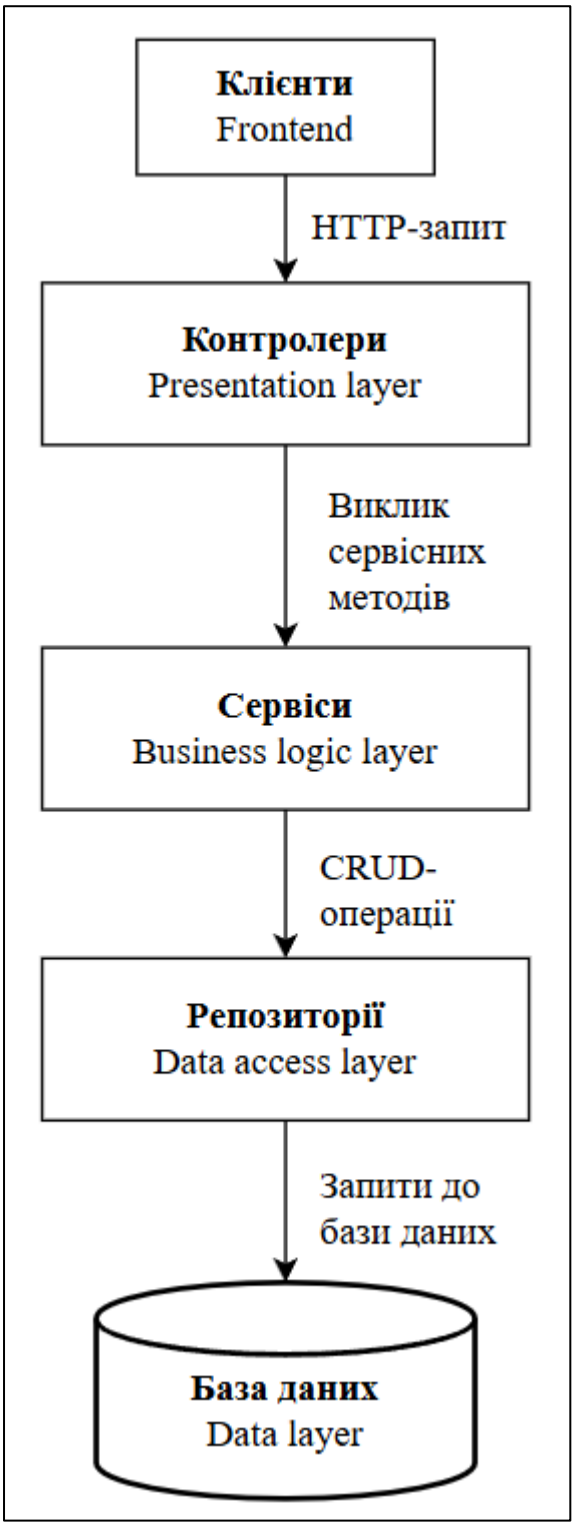


Рисунок Г.4 – Діаграма структурної взаємодії шарів архітектури застосунку

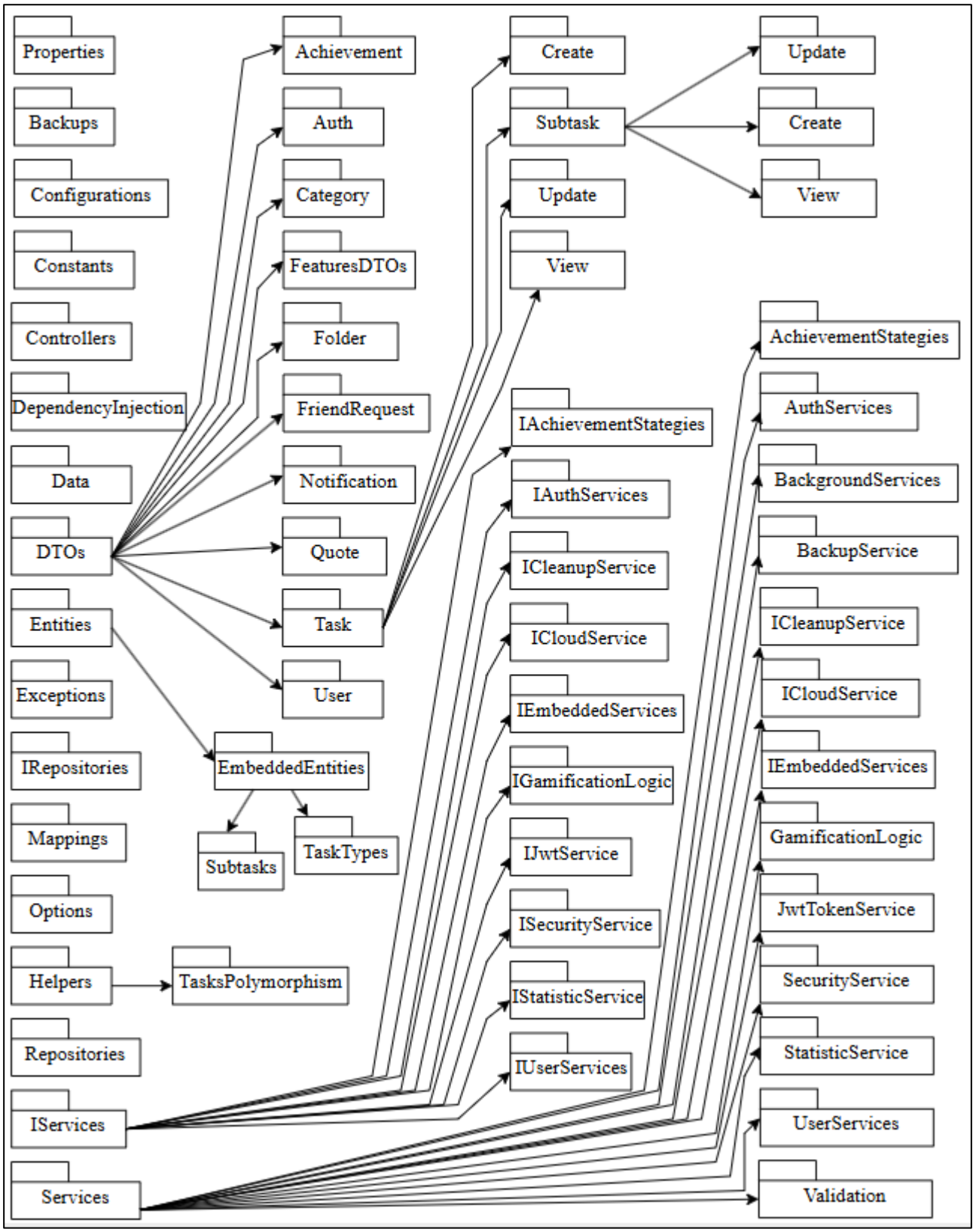


Рисунок Г.5 – Структура папок серверної частини програмної системи «Naviria»

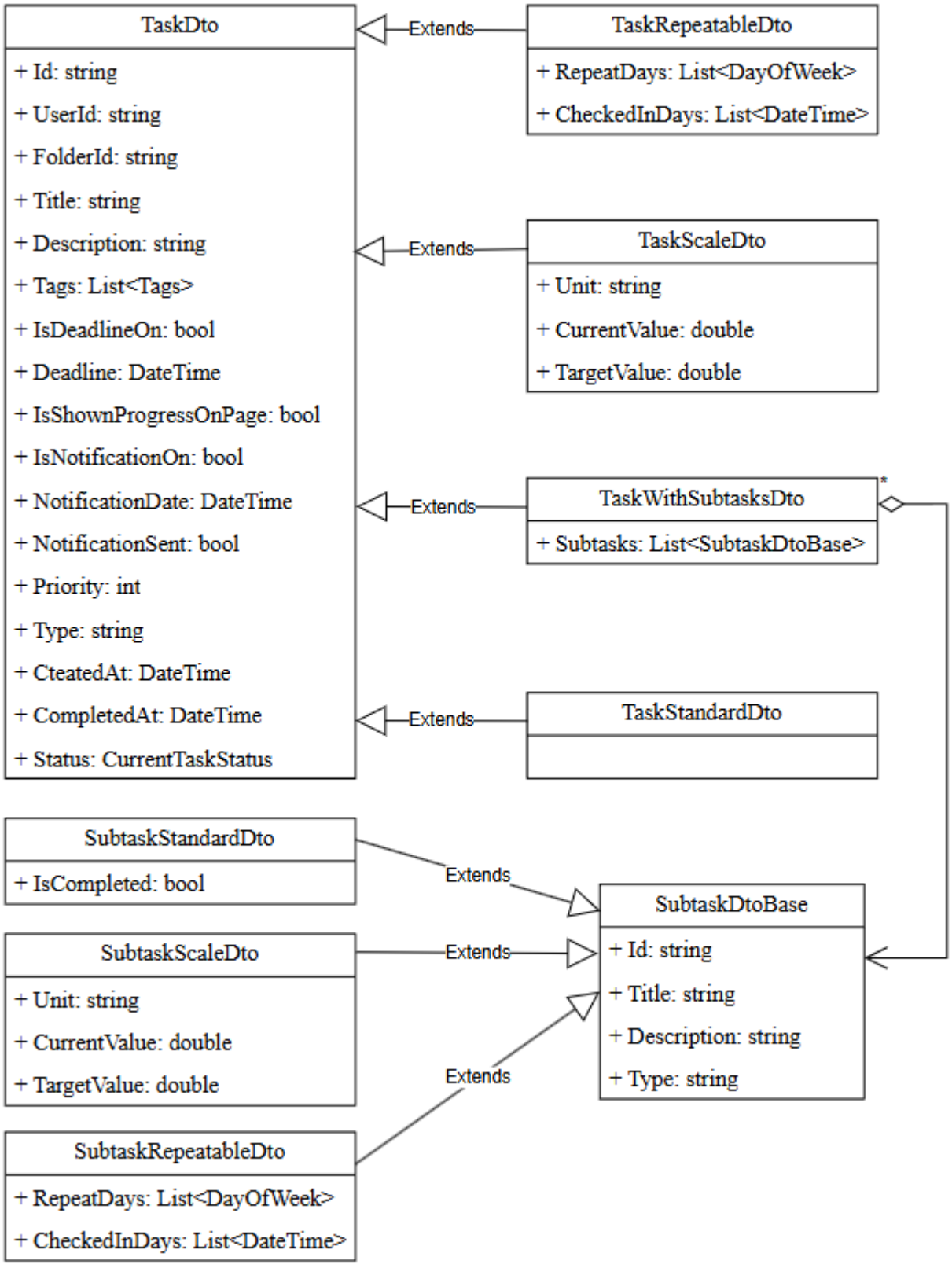


Рисунок Г.6 – Діаграма взаємодії класів задач та підзадач

ДОДАТОК Д

Програмний код

Д.1 Код класу ApplicationLayerServices.cs

```
using Microsoft.AspNetCore.Identity;
using NaviriaAPI.Entities;
using NaviriaAPI.IRepositories;
using NaviriaAPI.IServices;
using NaviriaAPI.IServices.IEmbeddedServices;
using NaviriaAPI.IServices.IGamificationLogic;
using NaviriaAPI.Repositories;
using NaviriaAPI.Services;
using NaviriaAPI.Services.GamificationLogic;
using NaviriaAPI.Services.User;
using NaviriaAPI.Services.AchievementStrategies;
using NaviriaAPI.Services.EmbeddedServices;
using NaviriaAPI.IServices.IUserServices;
using NaviriaAPI.Services.CleanupServices;
using NaviriaAPI.IServices.ICleanupServices;
using NaviriaAPI.Services.StatisticServices;
using NaviriaAPI.IServices.IStatisticServices;

namespace NaviriaAPI.DependencyInjection
{
    public static class ApplicationLayerServices
    {
        public static void AddApplicationServices(this IServiceCollection
services)
        {
            services.AddScoped<IAchievementRepository,
            AchievementRepository>();
            services.AddScoped<IAchievementService, AchievementService>();

            services.AddScoped<ICategoryRepository, CategoryRepository>();
            services.AddScoped<ICategoryService, CategoryService>();

            services.AddScoped<IAssistantChatService,
            AssistantChatService>();
            services.AddScoped<IAssistantChatRepository,
            AssistantChatRepository>();

            services.AddScoped<IFolderRepository, FolderRepository>();
            services.AddScoped<IFolderService, FolderService>();

            services.AddScoped<IFriendRequestRepository,
            FriendRequestRepository>();
            services.AddScoped<IFriendRequestService,
            FriendRequestService>();

            services.AddScoped<INotificationService,
            NotificationService>();
            services.AddScoped<INotificationRepository,
            NotificationRepository>();

            services.AddScoped<IQuoteRepository, QuoteRepository>();
```

```
            services.AddScoped<IQuoteService, QuoteService>();

            services.AddScoped<ITaskRepository, TaskRepository>();
            services.AddScoped<ITaskService, TaskService>();

            services.AddScoped<IUserRepository, UserRepository>();
            services.AddScoped<IUserService, UserService>();

            services.AddScoped<IPasswordHasher<UserEntity>,
            PasswordHasher<UserEntity>>();

            services.AddScoped<IAchievementManager, AchievementManager>();
            services.AddScoped<IAchievementGranter, AchievementGranter>();

            services.AddScoped<IAchievementStrategy,
            RegistrationAchievementStrategy>();
            services.AddScoped<IAchievementStrategy,
            FiveTasksInWeekAchievementStrategy>();
            services.AddScoped<IAchievementStrategy,
            FirstTaskCompletedAchievementStrategy>();
            services.AddScoped<IAchievementStrategy,
            LongTaskCompletedAchievementStrategy>();
            services.AddScoped<IAchievementStrategy,
            FiveFriendsAddedAchievementStrategy>();
            services.AddScoped<IAchievementStrategy,
            PhotoUploadedAchievementStrategy>();

            services.AddScoped<IAchievementCleanupService,
            AchievementCleanupService>();
            services.AddScoped<IUserCleanupService, UserCleanupService>();
            services.AddScoped<ICategoryCleanupService,
            CategoryCleanupService>();
            services.AddScoped<IFolderCleanupService,
            FolderCleanupService>();

            services.AddScoped<ISupportService, SupportService>();
            services.AddScoped<ISubtaskService, SubtaskService>();
            services.AddScoped<IUserSearchService, UserSearchService>();
            services.AddScoped<ITaskRewardService, TaskRewardService>();

            services.AddScoped<IGeneralStatisticService,
            GeneralStatisticsService>();
            services.AddScoped<IStatisticRepository,
            StatisticRepository>();
            services.AddScoped<ITaskStatisticService,
            RepeatableTaskStatisticService>();
            services.AddScoped<IStatisticsByCategoryService,
            StatisticsByCategoryService>();
            services.AddScoped<ITaskStatisticByDateService,
            TaskStatisticByDateService>();

            services.AddScoped<ILeaderboardService, LeaderboardService>();

        }
    }
}
```

Д.2 Код класу PolymorphicJsonConverter<TBase>

```
using System.Text.Json;
using System.Text.Json.Serialization;

namespace NaviriaAPI.Helpers.TasksPolymorphism
{
    public class PolymorphicJsonConverter<TBase> : JsonConverter<TBase>
    where TBase : class
    {
        private readonly string _typeProperty;
        private readonly Dictionary<string, Type> _typeMap;

        public PolymorphicJsonConverter(string typeProperty,
        Dictionary<string, Type> typeMap)
        {
            _typeProperty = typeProperty;
            _typeMap = typeMap;
        }

         public override TBase? Read(ref Utf8JsonReader reader, Type
        typeToConvert, JsonSerializerOptions options)
        {
            using var doc = JsonDocument.ParseValue(ref reader);
            var root = doc.RootElement;

            if (!root.TryGetProperty(_typeProperty, out var typeProp))
             throw new JsonException($"Missing '{_typeProperty}'
            property");

            var discriminator = typeProp.GetString();
            if (discriminator == null ||
            !_typeMap.TryGetValue(discriminator, out var targetType))
            throw new JsonException($"Unknown {_typeProperty}:
            {discriminator}");

            return (TBase?)JsonSerializer.Deserialize(root.GetRawText(),
            targetType, options);
        }

         public override void Write(Utf8JsonWriter writer, TBase value,
        JsonSerializerOptions options)
        {
        JsonSerializer.Serialize(writer, value, value.GetType(), options);
        }
    }
}
```

Д.3 SubtaskTypeMap.cs

```
using NaviriaAPI.Entities.EmbeddedEntities.Subtasks;
using NaviriaAPI.DTOs.TaskDtos;
using NaviriaAPI.DTOs.Task.Subtask.Create;
using NaviriaAPI.DTOs.Task.Subtask.View;
using NaviriaAPI.DTOs.Task.Subtask.Update;
```

```
namespace NaviriaAPI.Helpers
{
    public static class SubtaskTypeMap
    {
        public static Dictionary<string, Type> CreateMap => new()
        {
            { "standard", typeof(SubtaskStandardCreateDto) },
            { "repeatable", typeof(SubtaskRepeatableCreateDto) },
            { "scale", typeof(SubtaskScaleCreateDto) }
        };

        public static Dictionary<string, Type> UpdateMap => new()
        {
            { "standard", typeof(SubtaskStandardUpdateDto) },
            { "repeatable", typeof(SubtaskRepeatableUpdateDto) },
            { "scale", typeof(SubtaskScaleUpdateDto) }
        };

        public static Dictionary<string, Type> ReadMap => new()
        {
            { "standard", typeof(SubtaskStandardDto) },
            { "repeatable", typeof(SubtaskRepeatableDto) },
            { "scale", typeof(SubtaskScaleDto) }
        };

        public static Dictionary<string, Type> EntityMap => new()
        {
            { "standard", typeof(SubtaskStandard) },
            { "repeatable", typeof(SubtaskRepeatable) },
            { "scale", typeof(ScaleSubtask) }
        };
    }
}
```

Д.4 IAchievementStrategy.cs

```
using NaviriaAPI.Helpers;

namespace NaviriaAPI.IServices.IGamificationLogic
{
    public interface IAchievementStrategy
    {
        AchievementTrigger Trigger { get; }
        Task<IEnumerable<string>> GetAchievementIdsAsync(string userId,
         object? context = null);
    }

}
```

Д.5 AchievementGranter

```
using NaviriaAPI.Entities.EmbeddedEntities;
using NaviriaAPI.Exceptions;
using NaviriaAPI.IRepositories;
using NaviriaAPI.IServices.IGamificationLogic;
```

```
namespace NaviriaAPI.Services.GamificationLogic
{
    public class AchievementGranter : IAchievementGranter
    {
        private readonly IUserRepository _userRepository;
        private readonly IAchievementRepository _achievementRepository;
        private readonly ILogger<AchievementGranter> _logger;

        public AchievementGranter(
            IUserRepository userRepository,
            IAchievementRepository achievementRepository,
            ILogger<AchievementGranter> logger)
        {
            _userRepository = userRepository;
            _achievementRepository = achievementRepository;
            _logger = logger;
        }

        public async Task GiveAsync(string userId, string achievementId)
        {
            var user = await _userRepository.GetByIdAsync(userId);
            if (user == null) throw new NotFoundException("User not
            found");

            if (user.Achievements.Any(a => a.AchievementId ==
            achievementId))
                return;

            var achievement = await
           _achievementRepository.GetByIdAsync(achievementId);
            if (achievement == null) throw new
            NotFoundException("Achievement not found");

           _logger.LogInformation("Грант досягнення {AchievementId} для
            користувача {UserId}", achievementId, userId);

            user.Achievements.Add(new UserAchievementInfo
            {
                AchievementId = achievementId,
                ReceivedAt = DateTime.UtcNow,
                IsPointsReceived = false
            });

            await _userRepository.UpdateAsync(user);
        }
    }
}
```

Д.6 CloudinaryService

```
using CloudinaryDotNet.Actions;
using CloudinaryDotNet;
using NaviriaAPI.IRepositories;
using NaviriaAPI.IServices.ICloudStorage;

namespace NaviriaAPI.Services.CloudStorage
{
```

```
    public class CloudinaryService : ICloudinaryService
    {
        private readonly Cloudinary _cloudinary;

        public CloudinaryService(Cloudinary cloudinary)
        {
            _cloudinary = cloudinary;
        }

        public async Task<string> UploadImageAndGetUrlAsync(IFormFile file)
        {
            using var stream = file.OpenReadStream();
            var uploadParams = new ImageUploadParams
            {
                File = new FileDescription(file.FileName, stream),
                Folder = "users_photos",
                PublicId = Guid.NewGuid().ToString(),
                UseFilename = true,
                UniqueFilename = true,
                Overwrite = false
            };

            var uploadResult = await _cloudinary.UploadAsync(uploadParams);

            if (uploadResult.StatusCode == System.Net.HttpStatusCode.OK)
                return uploadResult.SecureUrl.ToString();

            throw new InvalidOperationException("Image upload failed.");
        }
    }
}
```

Д.7 GoogleAuthService

```
using Google.Apis.Auth;
using NaviriaAPI.Entities;
using NaviriaAPI.IRepositories;
using NaviriaAPI.IServices.IAuthServices;
using NaviriaAPI.IServices.IJwtService;
using Microsoft.AspNetCore.Identity;

namespace NaviriaAPI.Services.AuthServices
{
    public class GoogleAuthService : IGoogleAuthService
    {
        private readonly IUserRepository _userRepository;
        private readonly IJwtService _jwtService;
        private readonly string _googleClientId;

        public GoogleAuthService(
            IUserRepository userRepository,
            IJwtService jwtService,
            IConfiguration configuration)
        {
            _userRepository = userRepository;
            _jwtService = jwtService;
            _googleClientId =
```

```
configuration["Authentication:Google:WebClientId"];
        }

        public async Task<string> AuthenticateAsync(string idToken)
        {
            var payload = await
            GoogleJsonWebSignature.ValidateAsync(idToken, new
            GoogleJsonWebSignature.ValidationSettings
            {
                Audience = new[] { _googleClientId }
            });

            var email = payload.Email;
            var user = await _userRepository.GetByEmailAsync(email);

            if (user == null)
            {
                throw new ArgumentException("User with such email does not
                exist");
            }

            return _jwtService.GenerateUserToken(user);
        }
    }
}
```

Д.8 UserValidationService

```
using NaviriaAPI.DTOs.User;
using NaviriaAPI.IRepositories;
using System.ComponentModel.DataAnnotations;
using System.Text.RegularExpressions;

namespace NaviriaAPI.Services.Validation
{
    public class UserValidationService
    {
        private readonly IUserRepository _userRepository;

        public UserValidationService(IUserRepository userRepository)
        {
            _userRepository = userRepository;
        }

        public async Task ValidateCreateAsync(UserCreateDto dto)
        {
            if (await _userRepository.GetByEmailAsync(dto.Email) != null)
                throw new ArgumentException("User with this email already
                exists");

            if (await _userRepository.GetByNicknameAsync(dto.Nickname) !=
            null)
                throw new ArgumentException("User with this nickname
                already exists");

            ValidateFullName(dto.FullName, required: true);
            ValidateNickname(dto.Nickname, required: true);
```

```
        ValidateEmail(dto.Email, required: true);
        ValidatePassword(dto.Password, required: true);
        ValidateFutureMessage(dto.FutureMessage);

        var now = DateTime.UtcNow;
        if (dto.BirthDate > now)
        throw new ValidationException("Birth date cannot be in the
        future.");
        var age = now.Year - dto.BirthDate.Year;
        if (age < 18)
        throw new ValidationException("User must be at least 18 years
        old.");
        if (age > 120)
        throw new ValidationException("User cannot be older than 120
        years.");
    }

    public static void ValidatePatch(UserPatchDto dto)
    {
        ValidateFullName(dto.FullName);
        ValidateNickname(dto.Nickname);
        ValidateDescription(dto.Description);
        ValidateEmail(dto.Email);
        ValidatePassword(dto.Password);
        ValidatePoints(dto.Points);
        ValidateFutureMessage(dto.FutureMessage);
        ValidatePhoto(dto.Photo);
    }

    public static void ValidateUpdate(UserUpdateDto dto)
    {
        ValidateFullName(dto.FullName);
        ValidateNickname(dto.Nickname);
        ValidateDescription(dto.Description);
        ValidateEmail(dto.Email);
        ValidatePassword(dto.Password);
        ValidatePoints(dto.Points);
        ValidateFutureMessage(dto.FutureMessage);
        ValidatePhoto(dto.Photo);
    }

    private static void ValidateFullName(string? fullName, bool
     required = false)
    {
        if (string.IsNullOrEmpty(fullName))
        {
            if (required)
                throw new ValidationException("FullName is required.");
            return;
        }
        if (fullName.Length < 3 || fullName.Length > 50)
        throw new ValidationException("FullName must be between 3 and
        50 characters.");
        var regex = new Regex("^[a-zA-Za-яA-ЯёЁiIїЇєЄ\\s'-]{1,50}$");
        if (!regex.IsMatch(fullName))
                        throw new ValidationException("FullName
        contains invalid characters.");
    }
```

```
 private static void ValidateNickname(string? nickname, bool
 required = false)
{
    if (string.IsNullOrEmpty(nickname))
    {
        if (required)
            throw new ValidationException("Nickname is required.");
        return;
    }
    if (nickname.Length < 3 || nickname.Length > 20)
        throw new ValidationException("Nickname must be between 3
    and 20 characters.");
    var regex = new Regex("^[a-zA-Z0-9]+$");
    if (!regex.IsMatch(nickname))
    throw new ValidationException("Nickname must contain only
    letters and digits.");
}

private static void ValidateDescription(string? description)
{
    if (string.IsNullOrEmpty(description)) return;
    if (description.Length > 150)
     throw new ValidationException("Description must be less than
    150 characters.");
    var regex = new Regex("^[a-zA-Za-яA-ЯёЁiIiЇєЄ0-
    9.,!\\s]{0,150}$");
    if (!regex.IsMatch(description))
      throw new ValidationException("Description contains invalid
    characters.");
}

private static void ValidateEmail(string? email, bool required =
 false)
{
    if (string.IsNullOrEmpty(email))
    {
        if (required)
            throw new ValidationException("Email is required.");
        return;
    }
    if (email.Length > 100)
        throw new ValidationException("Email must be less than 100
    characters.");
    var regex = new Regex("^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-
    zA-Z]{2,}$");
    if (!regex.IsMatch(email))
        throw new ValidationException("Email format is invalid.");
    if (!new EmailAddressAttribute().IsValid(email))
        throw new ValidationException("Email is not valid.");
}

private static void ValidatePassword(string? password, bool
 required = false)
{
    if (string.IsNullOrEmpty(password))
    {
        if (required)
```

```
                throw new ValidationException("Password is required.");
            return;
        }
        if (password.Length < 8)
           throw new ValidationException("Password must be at least 8
        characters long.");
        var regex = new Regex("^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d).+$");
        if (!regex.IsMatch(password))
           throw new ValidationException("Password must contain at
         least one uppercase letter, one lowercase letter, and one
         digit.");
    }

    private static void ValidatePoints(int? points, bool required =
     false)
    {
        if (!points.HasValue)
        {
            if (required)
                throw new ValidationException("Points is required.");
            return;
        }
        if (points.Value < 0)
            throw new ValidationException("Points cannot be
     negative.");
    }

    private static void ValidateFutureMessage(string? futureMessage)
    {
        if (string.IsNullOrEmpty(futureMessage)) return;
        if (futureMessage.Length > 150)
            throw new ValidationException("FutureMessage must be less
        than 150 characters.");
        var regex = new Regex("^[a-zA-Za-яA-ЯёЁiIiЇєЄ0-
        9.,!\\s]{0,150}$");
        if (!regex.IsMatch(futureMessage))
            throw new ValidationException("FutureMessage contains
        invalid characters.");
    }

    private static void ValidatePhoto(string? photo)
    {
        if (string.IsNullOrEmpty(photo)) return;
        if (!Uri.TryCreate(photo, UriKind.Absolute, out _))
          throw new ValidationException("Photo must be a valid URL.");
    }
}

}
```

Д.9 MessageSecurityService

```
using NaviriaAPI.Exceptions;
using NaviriaAPI.IServices.ISecurityService;
using System.Text.RegularExpressions;

namespace NaviriaAPI.Services.SecurityServices
```

```
{
    public class MessageSecurityService : IMessageSecurityService
    {
        private readonly ILogger<MessageSecurityService> _logger;

        private static readonly string[] DangerousKeywords = new[]
        {
        "script", "alert", "onerror", "onload", "iframe", "src=",
        "SELECT", "INSERT", "DELETE", "DROP", "UPDATE", "--", ";", "/*",
       "*/", "OR 1=1", "UNION", "xp_cmdshell"
        };
        private static readonly Regex DangerousPattern = new Regex(

       @"(\b(SELECT|INSERT|DELETE|DROP|UPDATE|UNION|XP_CMDSHELL)\b|<script
       |</script>|--|\*\/|\/\*|OR\s+1=1)",
            RegexOptions.IgnoreCase | RegexOptions.Compiled
        );

         public MessageSecurityService(ILogger<MessageSecurityService>
         logger)
        {
            _logger = logger;
        }

        public void Validate(string userId, string message)
        {
            if (string.IsNullOrWhiteSpace(message))
                return;

            var matchedKeyword = DangerousKeywords
            .FirstOrDefault(k => message.Contains(k,
            StringComparison.OrdinalIgnoreCase));

            if (matchedKeyword != null)
            {
             LogAndThrow(userId, message, $"keyword match:
            {matchedKeyword}");
            }

            if (DangerousPattern.IsMatch(message))
            {
                LogAndThrow(userId, message, "regex match");
            }
        }

        private void LogAndThrow(string userId, string message, string
       reason)
        {
            _logger.LogWarning("♡ Suspicious message blocked from user
            {UserId}. Reason: {Reason}. Message: {Message}",
                userId, reason, message);

             throw new SuspiciousMessageException("Your message contains
             suspicious content and was blocked.");
        }
    }

}
```