

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Кафедра програмної інженерії

КУРСОВА РОБОТА
ПОЯСНЮВАЛЬНА ЗАПИСКА
з дисципліни “Об’єктно-орієнтоване програмування”
TELEGRAM BOT

Керівник проф.

Бондарєв В.М.

Студент гр. ПЗПІ-22-1

Тартаковський Д.В.

Комісія:

Проф. Бондарєв В.М.,

Ст. викл. Черепанова Ю.Ю.,

Ст. викл. Ляпота В.М.

Харків 2023

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра *програмної інженерії*

Рівень вищої освіти *перший (бакалаврський)*

Дисципліна *Об'єктно-орієнтоване програмування*

Спеціальність *121 Інженерія програмного забезпечення*

Освітня програма *Програмна інженерія*

Курс *1.*

Група *ПЗП-22-1.*

Семестр *2.*

ЗАВДАННЯ

на курсовий проект студента

Тартаковського Даніеля Володимировича

1 Тема проекту: Telegram Bot

2 Термін здачі студентом закінченого проекту: "16" - червня - 2023 р.

3 Вихідні дані до проекту:

Специфікація програми, методичні вказівки до виконання курсової роботи.

4 Зміст розрахунково-пояснювальної записки:

Вступ, опис вимог, проектування програми, інструкція користувача, висновки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапу	Термін виконання
1	Видача теми, узгодження і затвердження теми	13.02.2023 - 14.03.2023 р.
2	Формулювання вимог до програми	01.06.2023 - 2.06.2023 р.
3	Розробка моделей	05.06.2023 - 06.06.2023 р.
5	Розробка функцій	06.06.2023 - 09.06.2023 р.
8	Тестування і доопрацювання розробленої програмної системи	09.06.2023 - 17.06.2023 р.
9	Оформлення пояснювальної записки, додатків, графічного матеріалу	17.06.2023 - 18.06.2023 р.
10	Захист	19.06.2023 р.

Студент

Тартаковський Даніель Володимирович

Керівник

Бондарєв Володимир Михайлович

« 19 » червня 2023 р.

РЕФЕРАТ

Пояснювальна записка до курсової роботи: 26 с., 10 рис., 2 джерела.

КЛАС, КОЛЕКЦІЯ, МЕТОД, МОВА ПРОГРАМУВАННЯ C#,
КОРИСТУВАЧИ, ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ.

Метою роботи є розробка програми «Telegram Sticker Bot»,
надасть користувачам можливість створювати власні набори стікерів у
Telegram.

В результаті отриман Telegram Bot, який надає можливість
користувачам створювати набіри стікерів шляхом додавання зображень.
Бот також дозволяє редагувати ці набори стікерів, щоб користувачі
могли додавати нові стікери до своїх набірів.

В процесі розробки використано середовища Microsoft
Visual Studio 2022, платформи .NET 6.0, мова програмування C#.

ЗМІСТ

ВСТУП.....	5
1 ОПИС ВИМОГ	6
1.1 СЦЕНАРІЙ ВИКОРИСТАННЯ	6
1.2 ФУНКЦІОНАЛЬНІ ВИМОГИ	7
2 ПРОЕКТУВАННЯ ПРОГРАМИ.....	8
2.1 ОБ'ЄКТНА СТРУКТУРА ПРОГРАМИ.....	8
3 ІНСТРУКЦІЯ КОРИСТУВАЧА	10
3.1 ВСТАНОВЛЕННЯ ПРОГРАМИ	10
3.2 ФУНКЦІОНАЛ ПРОГРАМИ ТА ІНСТРУКЦІЯ ДО НЕЇ	10
3.2.1 Робота з «меню»	11
3.2.2 Робота з командою «/create»	12
3.2.3 Робота з командою «/add»	12
3.2.4 Робота з командою «/sets»	13
ВИСНОВОК	14
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	15
ДОДАТОК.....	16

ВСТУП

Метою цієї курсової роботи є створення «Telegram Sticker Bot»
Для цього були реалізовані наступні задачі:

- Створення наборів стікерів;
- Редагування наборів стікерів;
- Відображення всіх наборів стікерів, які належать користувачу;

В процесі розробки використано Microsoft Visual Studio,
платформу .NET 6.0, мову програмування C#.

1 ОПИС ВИМОГ

1.1 Сценарій використання

Сценарій 1. Створення Набора Стікерів

Передумова

Користувач натискає /start щоби розпочати роботу с ботом. Користувач обирає пункт «/create».

Основний сценарій

1. Програма запитує назву для набору.
2. Програма просить відправити стікер.
3. Набір стікерів успішно створено.

Додатковий сценарій

1. Якщо є якась помилка, користувача повертає на пункт «/create».

Сценарій 2. Додавання Стікерів у обраний користувачем набор

Передумова

Користувач вже створив набір стікерів. Користувач обирає пункт «/add».

Основний сценарій

1. Програма просить відправити стікер.
2. Програма просить відправити посилання на набір стікерів.
3. Програма просить відправити стікері які будуть додано до набору.

Додатковий сценарій

1. Якщо є якась помилка, користувача повертає на пункт «/add».

Сценарій 3. Відображення наборів стікерів

Передумова

Користувач вже створив набір стікерів. Користувач обирає пункт «/sets».

Основний сценарій

1. Програма відправляє набори стікерів які належать користувачу.

Додатковий сценарій

1. Якщо є якась помилка, користувача повертає на пункт «/sets».

1.2 Функціональні вимоги

Функція 1. Створення Набора Стікерів

Функція створення набору стікерів здійснюється через інтерфейс Telegram. Користувачеві буде запропоновано ввести назву набору та відправити першій стікер.

Функція 2. Додавання Стікерів у обраний користувачем набір

Функція додавання стікерів здійснюється через інтерфейс Telegram. Користувачеві буде запропоновано відправити посилання на набір стікерів. Після чого користувач має відправити стікері які будуть додані до цього набору.

Функція 3. Відображення наборів стікерів

Функція відображення наборів стікерів здійснюється через інтерфейс Telegram. Користувачеві будуть відображено всі його набори.

2 ПРОЕКТУВАННЯ ПРОГРАМИ

2.1 Об'єктна структура програми

Для реалізації проекту, який використовує Telegram Bot API, була обрана настільна архітектура з інтерфейсом (Console User Interface / Telegram Interface).

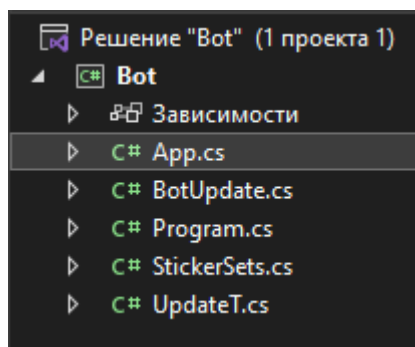


Рис. 2.1

Дані зберігатимуться у форматі JSON.

У проекті є такі класи:

1. Клас "App":

Представляє основну програму. Він відповідає за виконання головних функцій та керування іншими класами і структурами в проекті.

2. Клас "UpdateT":

Використовується для комунікації з JSON-даними, забезпечуючи обробку та передачу даних у цьому форматі.

3. Структура "BotUpdate":

```
namespace Bot
{
    Ссылка: 3
    public struct BotUpdate
    {
        public string text;
        public long id;
        public string? username;
    }
}
```

Рис. 2.2

Використовується для зберігання даних користувача.

4. Структура "StickerSets":

```
namespace Bot
{
    Ссылка: 6
    public struct StickerSets
    {
        public string stickerSetName;
        public long userId;
        public string? title;
    }
}
```

Рис. 2.3

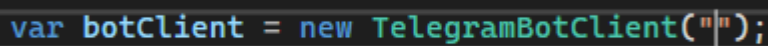
Використовується для зберігання даних про набори стікерів.

3 ІНСТРУКЦІЯ КОРИСТУВАЧА

3.1 Встановлення програми

Процедура встановлення програми:

1. Завантажити архів із програмою.
2. Розархівувати його у зручну директорію.
3. Відкрийте файл App.cs та знайдіть 32-й рядок, де знаходиться змінна `botClient = new TelegramBotClient("");`; Замініть порожній рядок на свій токен для підключення до Telegram Bot API.



```
var botClient = new TelegramBotClient("");
```

Рис. 3.1

4. Зібрати своє рішення, виконавши компіляцію проекту, та відкрити отриманий файл .exe для запуску програми.

3.2 Функціонал програми та інструкція до неї



Рис. 3.2 Telegram Interface

Користувач має натиснути «Розпочати».

3.2.1 Робота з «меню»



Рис 3.3

Користувач має натиснути на команду або написати її за допомогою символу "/" в чаті.

3.2.2 Робота з командою «/create»

Після того як користувач обирає опцію «/create». Користувачеві буде запропоновано ввести назву набору та відправити перший стікер.

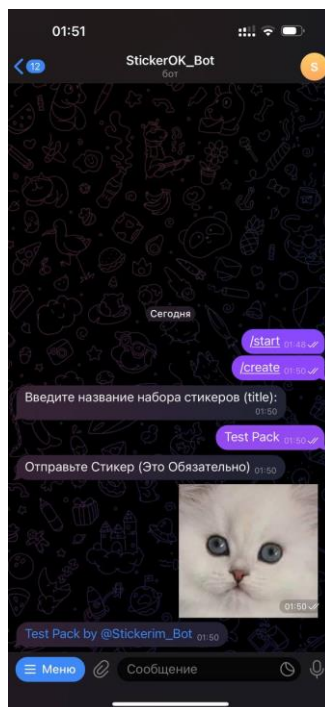


Рис 3.4 - /create

3.2.3 Робота з командою «/add»

Після того як користувач обирає опцію «/add». Користувачеві буде запропоновано відправити посилання на набір стікерів. Після чого користувач має відправити стікери які будуть додані до цього набору.

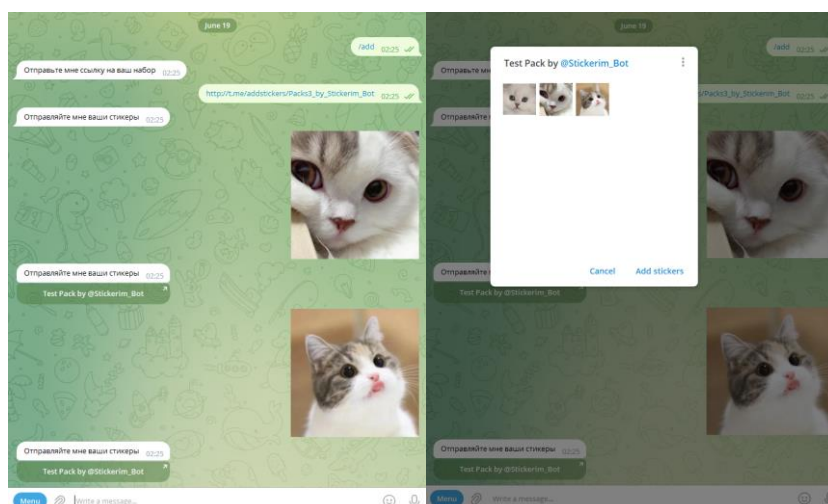


Рис 3.5 (а), Рис3.5 (б)

3.2.4 Робота з командою «/sets»

Після того як користувач обирає опцію «/sets». Користувачеві будуть відображено всі його набори.



Рис 3.6

ВИСНОВОК

У ході виконання курсової роботи на тему «Telegram Bot» була розроблена система на платформі .NET 6.0 з Telegram Interface для користувачів.

В результаті реалізації проекту було створено програму, яка використовує Telegram.Bot API для створення Telegram-бота з можливістю додавання стікерів та іншого функціоналу.

У подальшому проект можна розширити функціонал, оптимізувати методи, вдосконалити користувацький інтерфейс та перейти до використання бази даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Microsoft .Net Fundamentals documentation – документація Microsoft. URL: <https://learn.microsoft.com/en-us/dotnet/fundamentals/>
(дата звернення: 5.06.2023)

2. Telegram Bots Documentation - документація Telegram Bot.
URL: <https://telegrambots.github.io/book/>
(дата звернення: 5.06.2023)

ДОДАТОК

Код програми

```

using Newtonsoft.Json.Linq;
using Telegram.Bot;
using Telegram.Bot.Exceptions;
using Telegram.Bot.Polling;
using Telegram.Bot.Types;
using Telegram.Bot.Types.Enums;
using Telegram.Bot.Types.ReplyMarkups;

namespace Bot
{
    public class App
    {
        public async Task StartProgram()
        {
            List<BotUpdate> botUpdates = new List<BotUpdate>();
            List<StickerSets> stickerSets = new List<StickerSets>();
            List<BotCommand> commandsList = new List<BotCommand>();

            commandsList.Add(new BotCommand { Command = "start", Description =
"Начать использование бота" });
            commandsList.Add(new BotCommand { Command = "create", Description =
"Создать Стикер Пак" });
            commandsList.Add(new BotCommand { Command = "sets", Description =
"Ваши Стикер Паки" });
            commandsList.Add(new BotCommand { Command = "add", Description =
"Добавить Стикеров В Пак" });

            UpdateT updateT = new UpdateT();
            string filePath = "updates.json";
            string filePathStickerSet = "stickerSet.json";

            var botClient = new
TelegramBotClient("6125350302:AAECiOwG1HkjpWC07X6_ac7j-Flxug7KjMY");

            using CancellationTokenSource cts = new();
            var me = await botClient.GetMeAsync();
            // StartReceiving does not block the caller thread. Receiving is done
on the ThreadPool.
            ReceiverOptions receiverOptions = new()
            {

```

```

        AllowedUpdates = Array.Empty<UpdateType>() // receive all update
types except ChatMember related updates
    };

    botClient.StartReceiving(
        updateHandler: HandleUpdateAsync,
        pollingErrorHandler: HandlePollingErrorAsync,
        receiverOptions: receiverOptions,
        cancellation_token: cts.Token
    );

    Console.WriteLine($"Start listening for @{me.Username}");
    Console.ReadLine();

    cts.Cancel();

    async Task HandleUpdateAsync(ITelegramBotClient botClient, Update
update, Cancellation_token cancellation_token)
    {
        if (update.Message is not { } message)
            return;
        var chatId = update.Message.Chat.Id;

        if (update.Message.Text == "/Update")
        {
            BotCommand[] currentCommands = await
botClient.GetMyCommandsAsync();
            Dictionary<string, BotCommand> currentCommandsDict =
currentCommands.ToDictionary(cmd => cmd.Command);
            foreach (BotCommand newCommand in commandsList)
            {
                if (currentCommandsDict.TryGetValue(newCommand.Command,
out BotCommand existingCommand))
                {
                    if (existingCommand.Description !=
newCommand.Description)
                    {
                        existingCommand.Description =
newCommand.Description;
                    }
                }
                else
                {
                    currentCommands =
currentCommands.Append(newCommand).ToArray();
                }
            }
            await botClient.SetMyCommandsAsync(currentCommands);
        }
    }
}

```

```

var _botUpdate = new BotUpdate
{
    text = update.Message.Text,
    id = update.Message.Chat.Id,
    username = update.Message.Chat.Username
};

botUpdates.Add(_botUpdate);
await updateT.AppendToFileAsync(filePath, botUpdates);
botUpdates.Clear();
if (update.Message.Text != "Выход")
{
    switch (updateT.createSet)
    {
        case 0:
            switch (update.Message.Text)
            {
                case "/create":
                    await CreateSet(updateT.createSet);
                    break;
                case "/sets":
                    await StickerSets();
                    break;
                case "/add":
                    await AddStickersToSet(updateT.createSet);
                    break;
                default:
                    return;
            }
            break;
        case 1:
            await CreateSet(updateT.createSet);
            break;
        case 2:
            await CreateSet(updateT.createSet);
            break;
        case 3:
            await CreateSet(updateT.createSet);
            break;
        case 5:
            await AddStickersToSet(updateT.createSet);
            break;
        case 6:
            await AddStickersToSet(updateT.createSet);
            break;
        default:
            return;
    }
}

```

```

else
{
    updateT.createSet = 0;
    updateT.stickerSetNameGlobal = "";
    updateT.newTitle = "";
    return;
}

async Task CreateSet(int set)
{
    try
    {
        List<InputSticker> stickers = new List<InputSticker>();

        if (set == 0)
        {
            await botClient.SendTextMessageAsync(chatId, "Введите
название набора стикеров (title):");
            updateT.createSet = 1;
        }
        else if (set == 1)
        {
            updateT.newTitle = update.Message.Text;
            await botClient.SendTextMessageAsync(chatId,
"Отправьте Стикер (Это Обязательно)");
            updateT.createSet = 2;
        }
        else if (set == 2 && update.Message.Sticker != null)
        {
            string stickerSetName =
$"Packs{updateT.CountElementsInJsonFile(filePathStickerSet) +
1}_by_Stickerim_Bot";

            string newTitleUpdate = updateT.newTitle +
updateT.title;

            string stickerEmojis1 = "📎"; // Эмодзи для стикера
1
            stickers.Add(new
InputSticker(InputFile.FromFileId(update.Message.Sticker.FileId), new
List<string> { stickerEmojis1 }));
            await botClient.CreateNewStickerSetAsync(chatId,
stickerSetName, newTitleUpdate, stickers, StickerFormat.Static,
StickerType.Regular);

            Console.WriteLine("Набор стикеров успешно создан!");
            string markdownText =
$" [{newTitleUpdate}] ("http://t.me/addstickers/" + stickerSetName)";
            await botClient.SendTextMessageAsync(chatId,
markdownText, parseMode: ParseMode.Markdown);
            var _stickerSet = new StickerSets
            {
                stickerSetName = stickerSetName,

```

```

        userId = chatId,
        title = newTitleUpdate
    };

    stickerSets.Add(_stickerSet);
    await updateT.AppendToFileAsync(filePathStickerSet,
stickerSets);

    stickerSets.Clear();
    updateT.createSet = 0;
    updateT.newTitle = "";
}
else
{
    await botClient.SendTextMessageAsync(chatId,
"Повторите процедуру вписав /create - вы должны отправить 1 стикер");
    updateT.createSet = 0;
}
}
catch (Exception ex)
{
    Console.WriteLine($"Ошибка при получении наборов
стикеров: {ex.Message}");
}
}

async Task AddStickersToSet(int set)
{
    if (updateT.createSet == 0)
    {
        await botClient.SendTextMessageAsync(chatId, "Отправьте
мне ссылку на ваш набор");
        updateT.createSet = 5;
    }
    else if (updateT.createSet == 5)
    {
        int checker = 0;
        JArray filteredItems = await
updateT.FindItemsByUserIdFromFileAsync(filePathStickerSet, chatId);

        foreach (JObject item in filteredItems)
        {
            string stickerSetName =
item.Value<string>("stickerSetName");
            string stickerSetUserId =
item.Value<string>("userId");
            string stickerSetTitle = item.Value<string>("title");
            string stickerSetUrlHttp =
$"http://t.me/addstickers/{stickerSetName}";
            string stickerSetUrlHttps =
$"https://t.me/addstickers/{stickerSetName}";

```



```

        updateT.stickerSetTitleGlobal = "";
        updateT.createSet = 0;
    }
}
async Task StickerSets()
{
    try
    {
        JArray filteredItems = await
updateT.FindItemsByUserIdFromFileAsync(filePathStickerSet, chatId);
        var inlineKeyboardButtons = new
List<InlineKeyboardButton>();

        foreach (JObject item in filteredItems)
        {
            string stickerSetName =
item.Value<string>("stickerSetName");
            string stickerSetTitle = item.Value<string>("title");
            string stickerSetUrl =
$"http://t.me/addstickers/{stickerSetName}";
            var inlineKeyboardButton = new
InlineKeyboardButton(stickerSetTitle)
            {
                Url = stickerSetUrl,
            };

            inlineKeyboardButtons.Add(inlineKeyboardButton);
        }

        var inlineKeyboardMarkup = new
InlineKeyboardMarkup(inlineKeyboardButtons.Select(button => new[] { button
}).ToArray());
        await botClient.SendTextMessageAsync(chatId, "Sticker
Sets:", replyMarkup: inlineKeyboardMarkup);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Ошибка при получении наборов
стикеров: {ex.Message}");
    }
}

Task HandlePollingErrorAsync(ITelegramBotClient botClient, Exception
exception, CancellationToken cancellationToken)
{
    var ErrorMessage = exception switch
    {
        ApiRequestException apiRequestException

```

```

                => $"Telegram API
Error:\n[{apiRequestException.ErrorCode}]\n{apiRequestException.Message}",
                _ => exception.ToString()
            };

            Console.WriteLine(ErrorMessage);
            return Task.CompletedTask;
        }
    }
}

```

```

namespace Bot
{
    public struct StickerSets
    {
        public string stickerSetName;
        public long userId;
        public string? title;
    }
}

```

```

namespace Bot
{
    public struct BotUpdate
    {
        public string text;
        public long id;
        public string? username;
    }
}

```

```

namespace Bot
{
    class Program
    {
        static async Task Main(string[] args)
        {
            App app = new App();
            await app.StartProgram();
        }
    }
}

```

```

using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

```



```

namespace Bot
{
    public class UpdateT
    {
        public int createSet = 0;
        public string title = " by @Stickerim_Bot";
        public string newTitle = "";
        public string stickerSetNameGlobal = "";
        public string stickerSetTitleGlobal = "";
        public async Task AppendToFileAsync<T>(string filePath, List<T> items)
        {
            try
            {
                List<T> existingItems = await ReadFromFileAsync<T>(filePath);

                if (existingItems == null)
                {
                    existingItems = new List<T>();
                }

                existingItems.AddRange(items);
                await WriteToFileAsync(filePath, existingItems);
            }
            catch (Exception ex)
            {
                throw new InvalidOperationException($"Failed to append items to
the file. Exception: {ex.Message}");
            }
        }

        public async Task<List<T>> ReadFromFileAsync<T>(string filePath)
        {
            try
            {
                using StreamReader file = System.IO.File.OpenText(filePath);
                string json = await file.ReadToEndAsync();
                return JsonConvert.DeserializeObject<List<T>>(json);
            }
            catch (Exception ex)
            {
                throw new InvalidOperationException($"Failed to read items from
the file. Exception: {ex.Message}");
            }
        }

        public async Task WriteToFileAsync<T>(string filePath, List<T> items)
        {
            try
            {

```

```

        string json = JsonConvert.SerializeObject(items,
Formatting.Indented);

        using StreamWriter file = System.IO.File.CreateText(filePath);
        await file.WriteAsync(json);
    }
    catch (Exception ex)
    {
        throw new InvalidOperationException($"Failed to write items to
the file. Exception: {ex.Message}");
    }
}

public int CountElementsInJsonFile(string filePath)
{
    try
    {
        string json = System.IO.File.ReadAllText(filePath);
        List<object> items =
JsonConvert.DeserializeObject<List<object>>(json);
        return items.Count;
    }
    catch (FileNotFoundException ex)
    {
        throw new FileNotFoundException($"The specified file '{filePath}'
was not found. Exception: {ex.Message}");
    }
    catch (JsonSerializationException ex)
    {
        throw new InvalidOperationException($"Failed to deserialize JSON.
Exception: {ex.Message}");
    }
    catch (Exception ex)
    {
        throw new InvalidOperationException($"Failed to count elements in
JSON file. Exception: {ex.Message}");
    }
}

public async Task<JArray> FindItemsByUserIdFromFileAsync(string filePath,
long userId)
{
    try
    {
        List<StickerSets> stickerSets = await
ReadStickerSetsFromFileAsync(filePath);
        JArray jsonArray = JArray.FromObject(stickerSets);
        var filteredItems = jsonArray
            .Where(item => (long)item["userId"] == userId)
            .ToList();
        return new JArray(filteredItems);
    }
}

```

```

    }
    catch (Exception ex)
    {
        throw new InvalidOperationException($"Failed to find sticker sets
by user ID. Exception: {ex.Message}");
    }
}

public async Task<List<StickerSets>> ReadStickerSetsFromFileAsync(string
filePath)
{
    try
    {
        return await ReadFromFileAsync<StickerSets>(filePath);
    }
    catch (Exception ex)
    {
        throw new InvalidOperationException($"Failed to read sticker sets
from the file. Exception: {ex.Message}");
    }
}
}
}

```