МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**Практична Робота № 2**

з дисципліни « Архітектура проектування ПЗ .Net »

*назва дисципліни*

на тему: «ПАРАЛЕЛЬНІ ЗАДАЧИ ТА МЕРЕЖЕВА ВЗАЄМОДІЯ»

Виконав: студент 2 курсу групи № 621п

освітньої програми

121 інженерія програмного забезпечення

(шифр і назва ОП)

Чорний Р.С.

(прізвище й ініціали студента)

Прийняв: асистент каф. 603

Сьомочкін М.О.

(посада, науковий ступінь, прізвище й ініціали)

Кількість балів:

Харків – 2024

**ПОСТАНОВКА ЗАВДАННЯ**

Варіант – Бронювання екскурсій

Вивчити особливості розробки та програмування паралельних задач, мережеву взаємодію та закріпити навички об'єктно-орієнтованого програмування

Розробити програмний чат-бот для Telegram, використовуючи методи та засоби паралельного програмування платформи .Net. Бот повинен мати базовий набір функціональних можливостей для взаємодії з користувачами, включаючи відправлення повідомлень, обробку команд та використання асинхронних методів для ефективної роботи. Складність і функціональність бота визначається самостійно (див. табл. 2.1) і вибирається по одному з наступних напрямків:

– предметна область відповідає курсовому проекту з ООП (допускається реалізація як самостійної програми чат-бота, так і з інтеграцією безпосередньо у курсовий проект);

– предметна область визначається на основі даних з одного або декількох відкритих та загальнодоступних Web API [2 - 4]. Чат-бот фактично організує інтерфейс користувача в Telegram для інформації, яка отримується з вибраного Web API, наприклад, за допомогою методів класу HttpClient;

- Інші варіанти предметної галузі (після обговорення з викладачем).

1.У Telegram знайти спеціальний вбудований бот BotFather, в якому за допомогою команди /newbot створити власний чат-бот та отримати його API-Token. Зверніть увагу, що API-Token є секретом та не підлягає розголошенню!

2.Створити новий проект у середовищі розробки Visual Studio чи іншій IDE для .Net . Тип проекту може бути будь-яким.

3.Налаштувати проект для роботи з Telegram API (наприклад, через менеджер NuGet встановити пакет Telegram.Bot ) [ 1] .

4.Створити у програмі екземпляр класу TelegramBotClient з отриманим API-Token.

5.У створеному у п.4. об'єкті реалізувати та підключити обробники помилок та повідомлень бота.

6.Реалізувати код для обробки команд (наприклад, /start, /help тощо на власний розсуд), які керують логікою роботи чат-бота.

7.Реалізувати код надсилання повідомлень користувачу та обробки повідомлень, які надходять від користувача.

8.Використовувати асинхронні методи для мережної взаємодії та виконання тривалих операцій (наприклад, запити до стороннього API).

9.Протестувати бота на локальній машині та/або розгорнутого на хостингу.

**ВИКОНАННЯ ЗАВДАННЯ**

У Telegram знаходимо спеціальний вбудований бот BotFather, в якому за допомогою команди /newbot створюємо власного чат-бота та отримуємо його API-Token. (Рис. 1)

Изображение выглядит как текст, снимок экрана, дизайн

Автоматически созданное описание

Рисунок 1 – Створення чат бота та отримання його API-Token

Створюємо новий проект та налаштовуємо його для роботи з Telegram API (через менеджер NuGet встановлюємо пакет Telegram.Bot ). (Рис. 2)

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 2 – Встановлення пакету Telegram Bot

Створюємо у програмі екземпляр класу TelegramBotClient з отриманим API-Token

namespace APPZ\_Lab\_2

{

internal class BotService

{

TelegramBotClient botClient = new TelegramBotClient("ТОКЕН");

}

}

У створеному у п.4. об'єкті реалізуємо та підключаємо обробники помилок та повідомлень бота.

public async Task StartBotAsync()

{

var receiverOptions = new ReceiverOptions

{

AllowedUpdates = Array.Empty<UpdateType>()

};

botClient.StartReceiving(

HandleUpdateAsync,

HandleErrorAsync,

receiverOptions

);

var me = await botClient.GetMeAsync();

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

}

private async Task HandleUpdateAsync(ITelegramBotClient botClient, Update update, CancellationToken cancellationToken)

{

if (update.Type == UpdateType.Message && update.Message?.Text != null)

{

await HandleMessageAsync(botClient, update.Message);

}

}

private Task HandleErrorAsync(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;

}

Реалізуємо код для обробки команд /start, /help, /excursions, /book, /cancel

private async Task HandleMessageAsync(ITelegramBotClient botClient, Message message)

{

var chatId = message.Chat.Id;

var messageText = message.Text;

Console.WriteLine($"Received a '{messageText}' message in chat {chatId}.");

// Обробка команд

switch (messageText.Split(' ')[0])

{

case "/start":

await botClient.SendTextMessageAsync(chatId, "Welcome! Use /help to see available commands.");

break;

case "/help":

await botClient.SendTextMessageAsync(chatId, "Available commands: /start, /help, /excursions, /book, /cancel");

break;

case "/excursions":

await HandleExcursionsCommand(chatId);

break;

case "/book":

await HandleBookCommand(chatId, messageText);

break;

case "/cancel":

await HandleCancelCommand(chatId, messageText);

break;

default:

await botClient.SendTextMessageAsync(chatId, "Unknown command. Use /help to see available commands.");

break;

}

}

private async Task HandleExcursionsCommand(long chatId)

{

// Додаємо список екскурсій

var excursions = Program.excursions; // передбачається, що екскурсії знаходяться в Program

var response = "Available excursions:\n";

foreach (var excursion in excursions)

{

response += $"{excursion.Id}: {excursion.Name} - {excursion.Description}\n";

}

await botClient.SendTextMessageAsync(chatId, response);

}

private async Task HandleBookCommand(long chatId, string messageText)

{

// Логіка бронювання екскурсій

var parts = messageText.Split(' ');

if (parts.Length < 3)

{

await botClient.SendTextMessageAsync(chatId, "Usage: /book <excursion\_id> <participants>");

return;

}

if (int.TryParse(parts[1], out int excursionId) && int.TryParse(parts[2], out int participants))

{

var excursion = Program.excursions.FirstOrDefault(e => e.Id == excursionId);

if (excursion != null)

{

var userId = (int)chatId; // Використовуємо chatId як userId

excursion.Book(userId, participants);

await botClient.SendTextMessageAsync(chatId, $"Booked {participants} participants for excursion {excursion.Name}");

}

else

{

await botClient.SendTextMessageAsync(chatId, "Excursion not found.");

}

}

else

{

await botClient.SendTextMessageAsync(chatId, "Invalid input. Usage: /book <excursion\_id> <participants>");

}

}

private async Task HandleCancelCommand(long chatId, string messageText)

{

// Логіка скасування бронювання

var parts = messageText.Split(' ');

if (parts.Length < 2)

{

await botClient.SendTextMessageAsync(chatId, "Usage: /cancel <excursion\_id>");

return;

}

if (int.TryParse(parts[1], out int excursionId))

{

var excursion = Program.excursions.FirstOrDefault(e => e.Id == excursionId);

if (excursion != null)

{

var userId = (int)chatId; // Використовуємо chatId як userId

excursion.CancelBooking(userId);

await botClient.SendTextMessageAsync(chatId, $"Cancelled booking for excursion {excursion.Name}");

}

else

{

await botClient.SendTextMessageAsync(chatId, "Excursion not found.");

}

}

else

{

await botClient.SendTextMessageAsync(chatId, "Invalid input. Usage: /cancel <excursion\_id>");

}

Реалізуємо код надсилання повідомлень користувачу та обробки повідомлень, які надходять від користувача

private async Task HandleMessageAsync(ITelegramBotClient botClient, Message message)

{

var chatId = message.Chat.Id;

var messageText = message.Text;

Console.WriteLine($"Received a '{messageText}' message in chat {chatId}.");

// Обробка команд

switch (messageText.Split(' ')[0])

{

case "/start":

await botClient.SendTextMessageAsync(chatId, "Welcome! Use /help to see available commands.");

break;

case "/help":

await botClient.SendTextMessageAsync(chatId, "Available commands: /start, /help, /excursions, /book, /cancel");

break;

case "/excursions":

await HandleExcursionsCommand(chatId);

break;

case "/book":

await HandleBookCommand(chatId, messageText);

break;

case "/cancel":

await HandleCancelCommand(chatId, messageText);

break;

default:

await botClient.SendTextMessageAsync(chatId, "Unknown command. Use /help to see available commands.");

break;

}

}

private async Task HandleExcursionsCommand(long chatId)

{

// Додаємо список екскурсій

var excursions = Program.excursions; // передбачається, що екскурсії знаходяться в Program

var response = "Available excursions:\n";

foreach (var excursion in excursions)

{

response += $"{excursion.Id}: {excursion.Name} - {excursion.Description}\n";

}

await botClient.SendTextMessageAsync(chatId, response);

}

private async Task HandleBookCommand(long chatId, string messageText)

{

// Логіка бронювання екскурсій

var parts = messageText.Split(' ');

if (parts.Length < 3)

{

await botClient.SendTextMessageAsync(chatId, "Usage: /book <excursion\_id> <participants>");

return;

}

if (int.TryParse(parts[1], out int excursionId) && int.TryParse(parts[2], out int participants))

{

var excursion = Program.excursions.FirstOrDefault(e => e.Id == excursionId);

if (excursion != null)

{

var userId = (int)chatId; // Використовуємо chatId як userId

excursion.Book(userId, participants);

await botClient.SendTextMessageAsync(chatId, $"Booked {participants} participants for excursion {excursion.Name}");

}

else

{

await botClient.SendTextMessageAsync(chatId, "Excursion not found.");

}

}

else

{

await botClient.SendTextMessageAsync(chatId, "Invalid input. Usage: /book <excursion\_id> <participants>");

}

}

private async Task HandleCancelCommand(long chatId, string messageText)

{

// Логіка скасування бронювання

var parts = messageText.Split(' ');

if (parts.Length < 2)

{

await botClient.SendTextMessageAsync(chatId, "Usage: /cancel <excursion\_id>");

return;

}

if (int.TryParse(parts[1], out int excursionId))

{

var excursion = Program.excursions.FirstOrDefault(e => e.Id == excursionId);

if (excursion != null)

{

var userId = (int)chatId; // Використовуємо chatId як userId

excursion.CancelBooking(userId);

await botClient.SendTextMessageAsync(chatId, $"Cancelled booking for excursion {excursion.Name}");

}

else

{

await botClient.SendTextMessageAsync(chatId, "Excursion not found.");

}

}

else

{

await botClient.SendTextMessageAsync(chatId, "Invalid input. Usage: /cancel <excursion\_id>");

}

}

Додаємо асинхронні методи для мережної взаємодії та виконання тривалих операцій (запити до стороннього API OpenWeatherMap).

using System.Net.Http;

using System.Net.Http.Json;

using System.Threading.Tasks;

namespace APPZ\_Lab\_2

{

public class WeatherService

{

private readonly HttpClient \_httpClient;

private readonly string \_apiKey;

public WeatherService(string apiKey)

{

\_httpClient = new HttpClient();

\_apiKey = apiKey;

}

public async Task<string> GetWeatherAsync(string city)

{

var url = $"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={\_apiKey}&units=metric";

var response = await \_httpClient.GetFromJsonAsync<WeatherResponse>(url);

if (response != null)

{

return $"Weather in {city}: {response.Main.Temp}°C, {response.Weather[0].Description}";

}

return "Could not retrieve weather data.";

}

}

public class WeatherResponse

{

public WeatherInfo[] Weather { get; set; }

public MainInfo Main { get; set; }

}

public class WeatherInfo

{

public string Description { get; set; }

}

public class MainInfo

{

public float Temp { get; set; }

}

}

Тестуємо бота на локальній машині

Изображение выглядит как текст, снимок экрана, дизайн

Автоматически созданное описание

**ВИСНОВОК**

Під час виконання лабораторної роботи було розглянуто особливості розробки та програмування паралельних задач, мережеву взаємодію та закріплено навички об'єктно-орієнтованого програмування.

Розроблено програмного чат-бота для Telegram, використовуючи методи та засоби паралельного програмування платформи .Net