ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

Факультет комп'ютерно-інформаційних технологій та автоматизації

Кафедра прикладної математики і інформатики

«До захисту допущено»

Завідувачка кафедри ПМІ

Ольга ДМИТРІЄВА

«_7_» ____червня___ 2022 р.

Випускна кваліфікаційна робота

бакалавр

(освітній ступень)

на тему

«Розробка ігрового чат-боту «Вгадай футболіста за карткою»

зі спецчастиною

Розробка програмних модулів, інтерфейсу та бази даних засобами Python, SQLite.

Виконав: студент 4 курсу, групи ІПЗ-18

Спеціальності 121 Інженерія програмного забезпечення

Євген КОТЕРЕУ

(прізвище та ініціали) (підпис)

Керівник зав. каф. ПМІ, д.т.н., проф. Ольга ДМИТРІЄВА

(посада, науковий ступінь, вчене звання, ім'я та прізвище) (підпис)

Рецензент зав. каф. КІ, к.т.н., доц. Сергій КОВАЛЬОВ

(посада, науковий ступінь, вчене звання, ім'я та прізвище) (підпис)

Нормоконтроль:

Ірина НАЗАРОВА

(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Дата 24.05.2022

Студент

(підпис)

Луцьк – 2022 р.

ДВНЗ «Донецький національний технічний університет»
Факультет комп'ютерних наук і технологій
Кафедра прикладної математики та інформатики
Освітньо-кваліфікаційний рівень (освітній ступінь) бакалавр
Напрям підготовки (спеціальність) 121 Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Завідувачка кафедри
Ольга ДМИТРІЄВА
/_____/
"____" _____2022 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТУ

Євгену Котереу
1. Тема проєкту (роботи) Розробка ігрового чат-боту «Вгадай футболіста за карткою».
Спецчастина Розробка програмних модулів, інтерфейсу та бази даних засобами Python, SQLite.
Керівник проєкту (роботи) Ольга ДМИТРІЄВА, д.т.н, проф., зав. каф. ПМІ
(ім'я, прізвище, науковий ступінь, вчене звання, посада)
затверджені наказом від "06" травня 2022 року № 180
2. Строк подання студентом проєкту (роботи)_____
3. Вихідні дані до проєкту (роботи) результати науково-дослідної роботи, матеріали переддипломної практики.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1) аналіз предметної області і постановка завдання;
2) проєктування бази даних;
3) проєктування системи;
4) розробка інтерфейсу користувача;
5) розробка алгоритму веб-скрапінгу;
6) реалізація ігрового процесу;
7) тестування чат-боту.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
1) блок-схема розробленого алгоритму синтаксичного аналізу веб-сайту;
2) діаграма варіантів використання;
3) екранні форми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Підпис, дата	Підпис, дата

7. Дата видачі завдання 6 квітня 2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Об'єктний аналіз предметної області і постановка завдання	10.04.2022	
2.	Проєктування бази даних	13.04.2022	
3.	Проєктування системи	16.04.2022	
4.	Розробка інтерфейсу користувача	20.04.2022	
5.	Розробка алгоритму веб-скрапінгу	28.04.2022	
6.	Реалізація ігрового процесу	15.05.2022	
7.	Тестування чат-боту	26.05.2022	
8.	Оформлення пояснювальної записки	01.06.2022	
9.	Підготовка слайдів презентації	05.06.2022	
10.	Підготовка демонстраційної версії розробленого програмного продукту	10.06.2022	
11.	Написання доповіді	12.06.2022	

Студент _____ Євген КОТЕРЕУ
(ім'я, прізвище)

Керівник роботи _____ Ольга ДМИТРІЄВА
(підпис) (ім'я, прізвище)

АНОТАЦІЯ

Євген КОТЕРЕУ. Ігровий чат-бот «Вгадай футболіста за карткою» / Випускна кваліфікаційна робота на здобуття освітньо-кваліфікаційного рівня «бакалавр» за спеціальністю 121 Інженерія програмного забезпечення ДВНЗ ДонНТУ, Покровськ, Луцьк, 2022.

Об’єктом дослідження є процеси проектування, розробки та тестування ігрового чат-боту у месенджері Telegram.

Предметом дослідження є основні алгоритми і методи розробки ігрового чат-боту та бібліотека Aiogram для мови Python, яка реалізує механізми взаємодії з API-Telegram.

Метою роботи є розробка сценарію та логіки ігрового чат-боту «Вгадай футболіста за карткою» для месенджера Telegram засобами мови Python, бібліотеки Aiogram, бази даних SQLite.

Практичне значення розробки ігрового чат-боту полягає в створенні простої і цікавої гри з вільним доступом, яка не потребує інсталяції на пристрій, тобто не займає пам’ять девайсу.

Методи дослідження, які застосовано при створенні чат-боту та реалізації ігрового процесу, базуються на принципах об’єктно-орієнтованого програмування та модульного програмування, проєктування баз даних. Реалізація ігрового процесу здійснювалася на теоретичних знаннях теорії кінцевих автоматів, теорії алгоритмів, теорії обробки зображень.

Ключові слова: чат-бот, Python, Aiogram, SQLite3, машина станів, веб-скрапінг.

Список публікацій здобувача:

1. Котереу Є. І. Розробка ігрового чат-боту для футбольних вболівальників / Є. І. Котереу. – Одеса: зб. ОНТУ, 2022 (у друці).

ANNOTATION

Yevhen KOTEREU. Game chatbot "Guess the football player by card" / Graduate qualification work for bachelor's degree in 121 Software Engineering of DonNTU, Pokrovsk, Lutsk, 2022.

The object of the study is the process of designing, developing and testing a game chatbot in Telegram messenger.

The subject of the study is the basic algorithms and methods of developing a game chatbot and the Aiogram library for the Python language, which implements mechanisms for interacting with API-Telegram.

The purpose of the work is to develop the script and logic of the game chatbot "Guess the football player by card" for Telegram messenger using the language Python, Aiogram library, SQLite database.

The practical importance of developing a game chatbot is to create a simple and interesting game with free access that does not require installation on the device, that is, does not take up memory of the device.

The research methods used in the creation of a chatbot and the implementation of the gameplay are based on the principles of object-oriented programming and modular programming, database design. The implementation of the game process was carried out on theoretical knowledge of finite-state machine theory, theory of algorithms, image processing theory.

Keywords: chatbot, Python, Aiogram, SQLite3, state machine, web scraping.

List of applicant`s publications:

1. Kotereu Y. Development of a game chatbot for football fans / Y. Kotereu. – Odesa: coll. ONTU, 2022 – print.

ЗМІСТ

ВСТУП

Велику популярність набули не складні ігри для мобільних пристроїв. Під час поїздки в громадському транспорті чи у вільну хвилину багато хто грає на мобільному телефоні у якусь просту та недовготривалу гру чи переглядає соціальні мережі.

Сучасні месенджери та соціальні мережі – мають такий функціонал як віртуальний співбесідник, так званий чат-бот. Чат-бот це програма, яка імітує взаємодію з реальним користувачем, задовольняючи потреби людини, яка використовую цей бот. Кожен день створюється багато різноманітних ботів, з обширною тематикою.

Певну популярність мають й ігрові чат-боти. Оскільки вони мають перевагу над ігровими додатками, їх не потрібно встановлювати на смартфон, а отже, вони не використовують ресурси телефону.

Враховуючи вищевказане обґрунтування темою кваліфікаційної роботи обрано розробку ігрового чат-боту «Вгадай футболіста за карткою».

Об’єкт дослідження – процес проектування та розробки ігрового чат-боту для месенджера Telegram.

Мета роботи – розробка ігрового чат-боту з базою даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОСНОВНІ ЗАВДАННЯ РОБОТИ

1.1 Обґрунтування теми роботи та її актуальність

Люди проводять багато часу у смартфоні. Месенджери, мобільні ігри та інші розважальні додатки займають значну частину часу в житті людей. Поїздка в громадському транспорті, вільні декілька хвилин на роботі чи в навчальному закладі здебільшого супроводжуються переглядом соціальних мереж, месенджерів, грою у мобільні ігри.

Тож популярність ігор на мобільних телефонах значно зросла, а отже, й кількість вакансій на посаду розробника мобільних ігор також.

В якості теми кваліфікаційної роботи було обрано розробку ігрового чат-боту «Вгадай футболіста за карткою» для Telegram.

1.2 Мета роботи та практичне значення

Метою роботи є розробка сценарію та логіки ігрового чат-боту «Вгадай футболіста за карткою» для месенджера Telegram засобами мови Python [1], бібліотеки Aiogram [2], бази даних SQLite [3].

Практичне значення розробки ігрового чат-боту полягає в створенні простої і цікавої гри з вільним доступом, яка не потребує інсталяції на пристрій, тобто не займає пам’ять девайсу.

1.3 Об’єкт, предмет та методи дослідження

Об’єктом дослідження є процеси проектування, розробки та тестування ігрового чат-боту у менеджері Telegram.

Предметом дослідження є основні алгоритми і методи розробки ігрового чат-боту та бібліотека Aiogram для мови Python, яка реалізує механізми взаємодії з API-Telegram.

Методи дослідження, які застосовано при створенні чат-боту та реалізації ігрового процесу, базуються на принципах об’єктно-орієнтованого програмування [4] та модульного програмування [5], проєктування баз даних. Реалізація ігрового процесу здійснювалася на теоретичних знаннях теорії кінцевих автоматів [6], теорії алгоритмів [7], теорії обробки зображень [8].

1.4 Сучасні підходи до реалізації ігрових чат-ботів

У месенджері Telegram існує декілька чат-ботів та каналів, у яких пропонується вгадати футболістів. Серед наявних чат-ботів працює лише один англомовний. Чат-бот пропонує вгадувати за фотографією або кар’єрою гравця.

Існуючі канали ведуться вручну, там створюються опитувальники, які охоплюють футбольну тематику загалом. Не ведеться ніяка статистика гравців. Нові завдання з’являються не регулярно, їх поява залежить від бажання та вільного часу автора каналу.

Приклад роботи англомовного чат-боту показаний на рисунках 1.1-1.2.

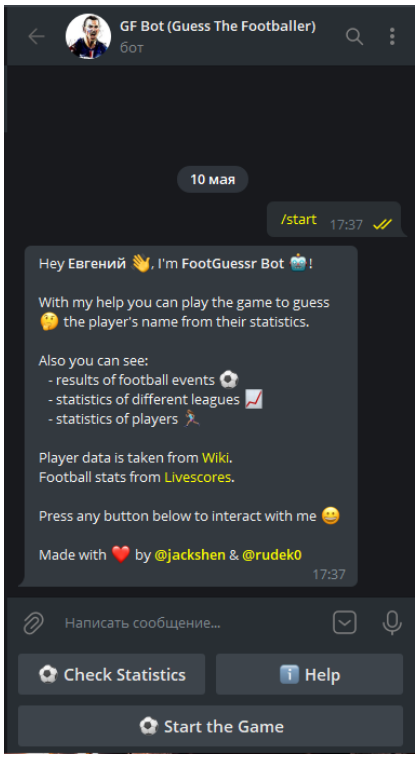

Рисунок 1.1 – Англомовний бот запущено, взято з [9]

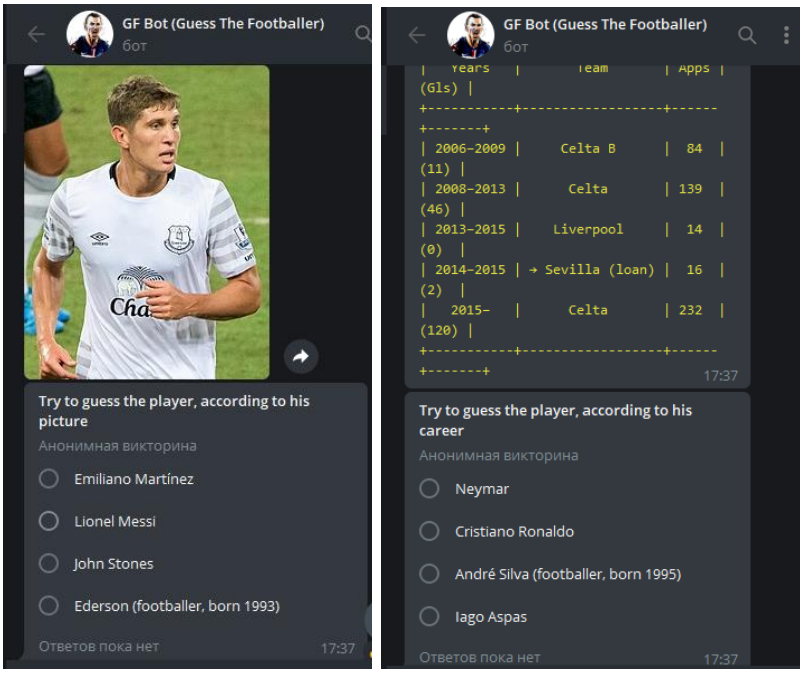

Рисунок 1.2 – Режим вгадування за фотографією та кар’єрою футболіста, взято з [9]

Приклад роботи каналу з опитувальниками зображено на рисунку 1.3.

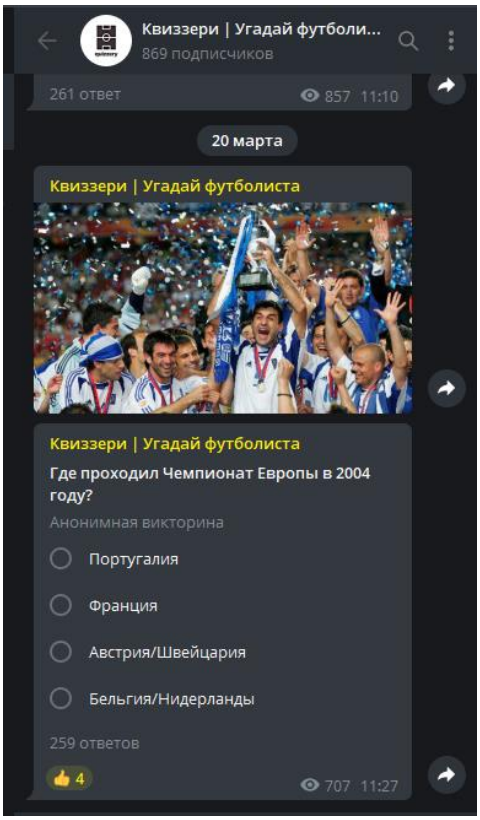

Рисунок 1.3 – Telegram-канал з опитувальниками, взято з [10]

1.5 Обґрунтування застосування інструментарію та технологій для розробки програмного продукту

Реалізація ігрового чат-боту передбачає необхідність зберігання інформації про футболістів та користувачів чат-боту.

Функціонал чат-боту передбачає упорядкований збір інформації про футболістів, а саме зображення карток та інформації для підказок: рейтинг, національність, клуб та позиція. Для впорядкованого збору інформації необхідно застосувати технологію веб-скрапінгу [11].

Отриману інформацію для підказок про кожного футболіста слід зберігати.

Також функціонал боту передбачає ведення статистики користувачів, а саме – кількість набраних ними очок.

Враховуючи ці вимоги, необхідно спроєктувати базу даних з таблицями футболістів та користувачів боту.

Зображення карток також необхідно зберігати. Для цього слід виділити невелику кількість пам’яті на сервері, де буде розташовано чат-бот. Так, для зберігання 140 карток необхідно 13 МБ пам’яті.

Безпосередньо ігровий процес потребує запам’ятовування дій користувача, кількість набраних очок, яка залежить від кількості використаних підказок, вгадав чи не вгадав користувач футболіста. Для вирішення цієї задачі слід застосувати патерн проєктування [12] «Стан» [13] та, відповідно, розробити машину станів. Реакція системи на зміну стану та певні вхідні дані реалізовуватиметься за допомогою обробників подій.

Виходячи з інформації, яка викладена вище, програмну систему можна розділити на декілька модулів за принципом модульного програмування. У процесі розробки програмних продуктів, на етапі проєктування будуються UML-діаграми [14] та блок-схеми. Діаграми спрощують написання коду, за допомогою діаграм можна відобразити внутрішню архітектуру системи, взаємозв’язки та шляхи обміну даними між модулями і функціями програми.

1.5.1 Мова UML у проєктуванні програмних продуктів

При проєктуванні програмних продуктів застосовується мова UML. UML – стандартний інструмент для візуалізації, специфікації, конструювання та документування компонентів програмних систем [14].

Переваги мови UML:

– об'єктно-орієнтована мова;

– UML дозволяє описати систему з будь-якої точки зору;

– дозволяє описати різні аспекти системи;

– стрімко розвивається та широко застосовується;

– мову, можна використовувати для опису процесів в багатьох галузях життя.

Для створення моделі системи за допомогою мови UML використовується два основних поняття: сутності та відношення.

Існує 4 типи сутностей:

а) структурні:

1) клас;
2) інтерфейс;
3) компонент;
4) варіант використання;
5) кооперація;
6) вузол;

б) поведінкові:

1) взаємодія;
2) стан;

в) сутності, які групують;

г) анотації.

Кожна сутність має власне графічне представлення.

Відношення демонструють різні зв’язки між сутностями. У UML представлені такі типі відносин:

– залежності;

– асоціація;

– узагальнення;

– реалізація.

1.5.2 Модульне програмування

Модульне програмування – розбиття програмного коду на окремі файли – модулі. Ці модулі можна в подальшому використовувати й у інших програмах. Модулі у мові python це звичайні файли з розширенням .py.

Коли створено декілька модулів, які мають схожу тематику, їх об’єднують в пакети – для більш зручного та структурованого використанням, зберігання та поширення.

Пакет – каталог, який містить багато модулів і файл __init__.py, необхідний для створення цього пакету и коректної взаємодії основної програми з модулями в поточному пакеті. На рисунку 1.4 продемонстровано приклад пакету з кількома модулями.

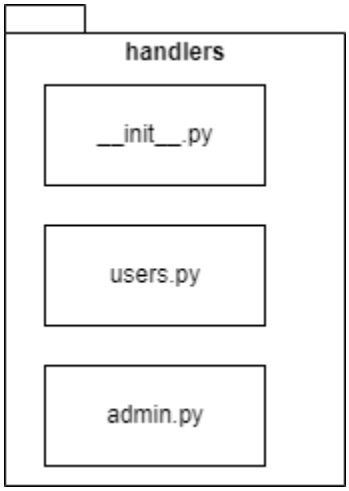

Рисунок 1.4 – Пакет з кількома модулями на мові python

1.5.3 Веб-скрапінг

Інформацію про футболістів та зображення карток з гри FIFA 22, які використовує бот в ігровому процесі необхідно упорядковано зібрати.

Для виконання цієї задачі слід розробити алгоритм веб-скрапінгу, який спочатку перетворює розмітку сторінки у дерево синтаксичного розбору, а потім витягує потрібні дані.

Веб-скрапінг – технологія отримання даних зі сторінок веб-ресурсів. Зазвичай цей процес автоматизують. Отримані дані структурують та впорядковують для облегшення подальшої роботи з даними.

1.5.4 Розмітка документів

Для створення сторінок в Інтернеті використовується мова гіпертекстової розмітки HTML [15]. Основними інструментами мови є теги.

Теги – елемент мови розмітки, зазвичай є парним – початковий та кінцевий. Тег інформує браузер про те, яким чином необхідно відобразити вміст відповідного тегу.

Теги мають властивості – атрибути. Атрибути слугують для додаткового форматування елементів, що містяться в середині тегу. Записуються атрибути парою: ім’я-значення.

Оскільки на веб-сайті багато однакових тегів, для більш індивідуального дизайну й роботи з елементами сайту використовують атрибут class, який дає можливість працювати окремо з різними елементами, які створені одним й тим же тегом.

Наприклад, на рисунку 1.5 червоною рамкою виділено два різних елементи сторінки, які створені за допомогою тегу . Якщо винести стильове оформлення сторінки в окремий файл, то саме завдяки атрибуту class браузер би зміг відрізняти різне стильове оформлення цих двох елементів.

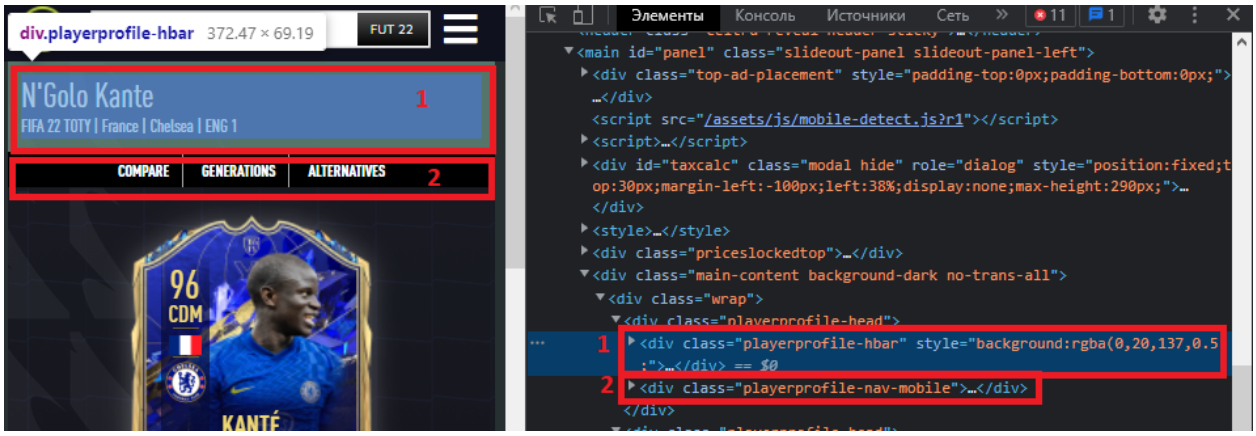

Рисунок 1.5 – Приклад двох елементів під тегом з атрибутами class

1.5.5 Синтаксичне дерево

Дерево представляє ієрерхічну структуру вузлів на кількох рівнях [16]. Усі вузли, окрім останнього, пов’язані з кількома на наступному рівні, і з одним на попередньому.

Перший рівень містить лише один вузол – корінь, вузли останнього рівня – листи. Кущ вузла – множина підлеглих йому вузлів нижніх рівнів.

1.5.6 Патерн проєктування «Стан»

«Стан» – патерн проєктування, який дозволяє об’єкту змінювати поведінку в залежності від стану, в якому він перебуває.

Програма знаходиться в одному з кількох станів, переходячи час від часу в інший. Набор станів та переходів між ними кінцевий та заздалегідь визначений. Тобто, реалізується модель кінцевого автомату.

На рисунку 1.6 приведено приклад застосування патерну до об’єкта «Документ».

Рисунок 1.6 – Стани документа та переходи між ними, взято з [13]

Використання цього патерну має ряд переваг [13]:

– позбавляє код від великої кількості умовних операторів;
– концентрує в одному місці код, пов’язаних з певним станом;
– спрощує код контексту.

Патерн має й недолік – безпідставне ускладнення коду, якщо станів мало, й змінюються вони рідко.

1.5.7 Кінцевий автомат

Кінцевий автомат (машина станів) – різновид автомата-абстракції. Метою створення кінцевого автомату є опис зміни стану об’єкта, залежно від поточного стану та вхідних даних. Автомат є дискретним, тобто має обмежену кількість станів, має початковий та кінцевий стани.

Для реалізації ігрового процесу необхідний автомат, який відноситься до перетворювачів.

Перетворювач (трансдуктор) – автомат, який формує вихідні дані, залежно від даних, отриманих на вході, та станів.

Для організації коректної роботи кінцевого автомата, він має зберігати деякі ігрові дані, тобто мати пам’ять. У якості пам’яті використовуватиметься дискретна структура – словник. Словник – пара «ключ» – «значення».

Кінцевий автомат реалізовуватиметься за принципами об’єктно-орієнтованого програмування. Приклад простої машини станів «турнікет» показано на рисунку 1.7.

Рисунок 1.7 – Машина станів «турнікет»

1.5.8 Об’єктно-орієнтовне програмування

ООП – методологія програмування, при якій програма написана у вигляді об’єктів, що взаємодіють між собою [17]. Об’єкти є екземплярами класів, а класи утворюють ієрархію успадкування.

Клас – абстрактний тип даних з можливо частковою реалізацією. Клас складається з набору полів (атрибутів) та методів.

Абстрактний клас описує загальну структуру об’єктів, які мають схожі властивості. А клас нащадок, який успадковує ці властивості від батьківського, більш конкретизовано описує об’єкт. Наприклад, клас фігур – батьківський, прямокутник – клас-нащадок (рис. 1.8).

Клас має глобальні змінні – поля та функції – методи класу. Поля – описують властивості об’єкта класу, а методи визначають поведінку.

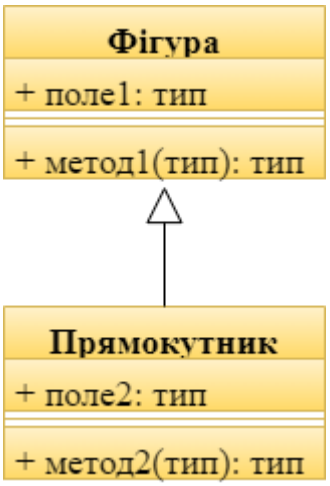

Рисунок 1.8 – Приклад наслідування

1.6 Основні завдання роботи

Виходячи з теми та сформованої мети, в роботі будуть поставлені такі завдання:

– упорядкований збір інформації про футболістів;

– зберігання інформації у спроєктованій базі даних;

– використання отриманої інформації про футболістів в ігровому процесі;

– обробка зображень при зборі інформації;

– створення машини станів для реалізації ігрового процесу.

1.7 Висновки за розділом

У цьому розділі було обґрунтовано вибір теми кваліфікаційної роботи, визначено її мету.

Також було проведено аналіз конкурентів, за результатами якого було виявлено лише один англомовний чат-бот, який схожий за тематикою, та декілька телеграм каналів, які охоплюють футбольну тематику загалом. Такі канали не функціонують постійно, на відміну від чат-боту.

У розділі описані технічні особливості реалізації роботи, системні вимого по необхідному об’єму пам’яті та необхідні теоретичні знання та положення для вирішення поставлених задач.

Основними задачами дослідження є:

– реалізація логіки ігрового процесу з використанням патерну проєктування «Стан» та за допомогою кінцевого автомату з пам’яттю;

– система ведення статистики набраних очок користувачами за допомогою бази даних;

– алгоритм веб-скрапінгу для зчитування інформації про футболістів та завантаження зображень карток з інтернет-джерела;

– реалізація загальнодоступних команд та команд для адміністратора.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Розробка архітектури програмної системи

У месенджері Telegram необхідно знайти бота з назвою BotFather, де потрібно зареєструвати нового бота, дати йому назву та username – унікальний нікнейм. Після успішного створення, BotFather надасть токен, за допомогою якого можна керувати з ботом.

Для взаємодії з Telegram Bot API слід обрати одну з існуючих бібліотек для мови Python. Найбільш популярні серед них – python-telegram-bot [18] та aiogram.

Створення екземпляра боту та його запуск виокремлені у різні модулі.

Бот матиме декілька команд загальних та дві команди для адміністратора:

– /start – запустити бота;

– /help – правила гри;

– /game – почати гру;

– /record – переглянути ТОП-3 та власну кількість очок;

– /download – команда адміністратора для завантаження карток до БД;

– /delete – команда адміністратора для очищення таблиці з футболістами.

Окремий модуль для обробки команда адміністратора з перевіркою чи є користувач адміністратором. Модуль для обробки команд і кнопок усіх користувачів чат-боту.

При першому запуску бот заносить нового користувача до бази даних у відповідну таблицю.

Командою /download адміністратор запускає модуль веб-скрапінгу, який опрацьовує певну кількість сторінок сайту https://www.futwiz.com/ru, де містяться картки, які модуль завантажує на комп’ютер чи сервер, на якому

розгорнута робота системи, та інформацію про футболістів, яку модуль збирає до БД функцією з модулю для роботи з базою даних.

Відповідно, команда /delete видаляє записи з таблиці футболістів та зображення карток, які збережені.

На рисунку 2.1 показана діаграма варіантів використання [19] для актора «Адміністратор». Діаграми діяльності [20] для актора представлені на рисунках 2.2 і 2.3.

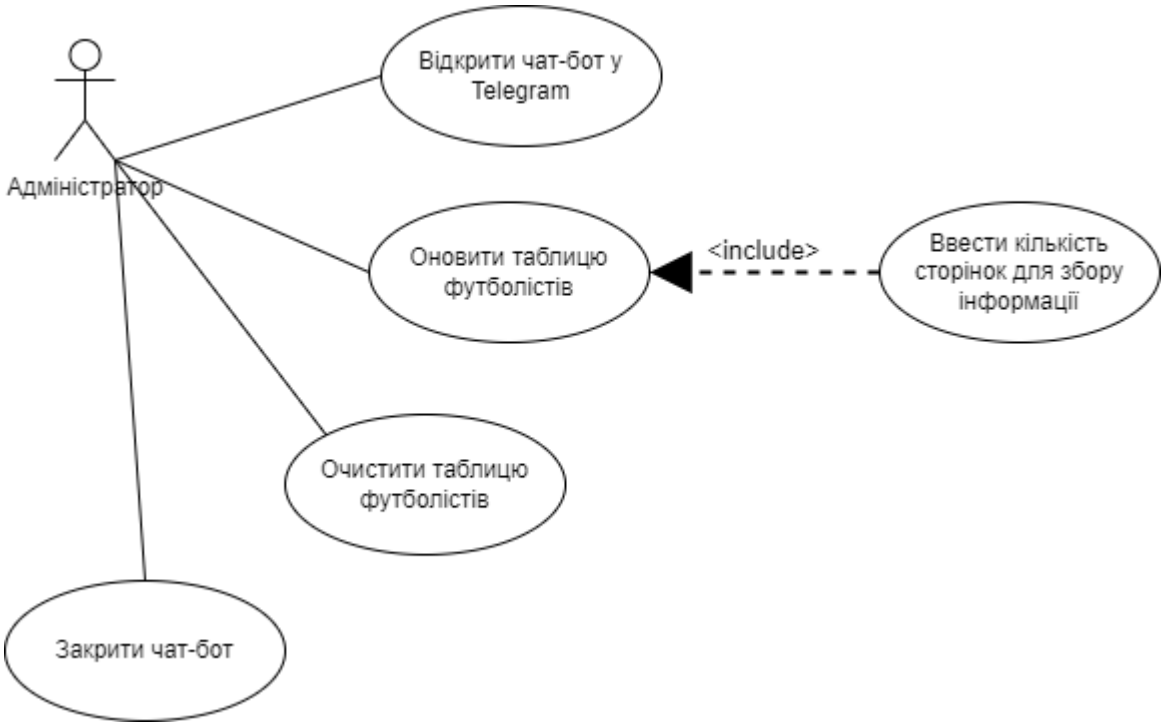

Рисунок 2.1 – Діаграма варіантів використання для актора «Адміністратор»

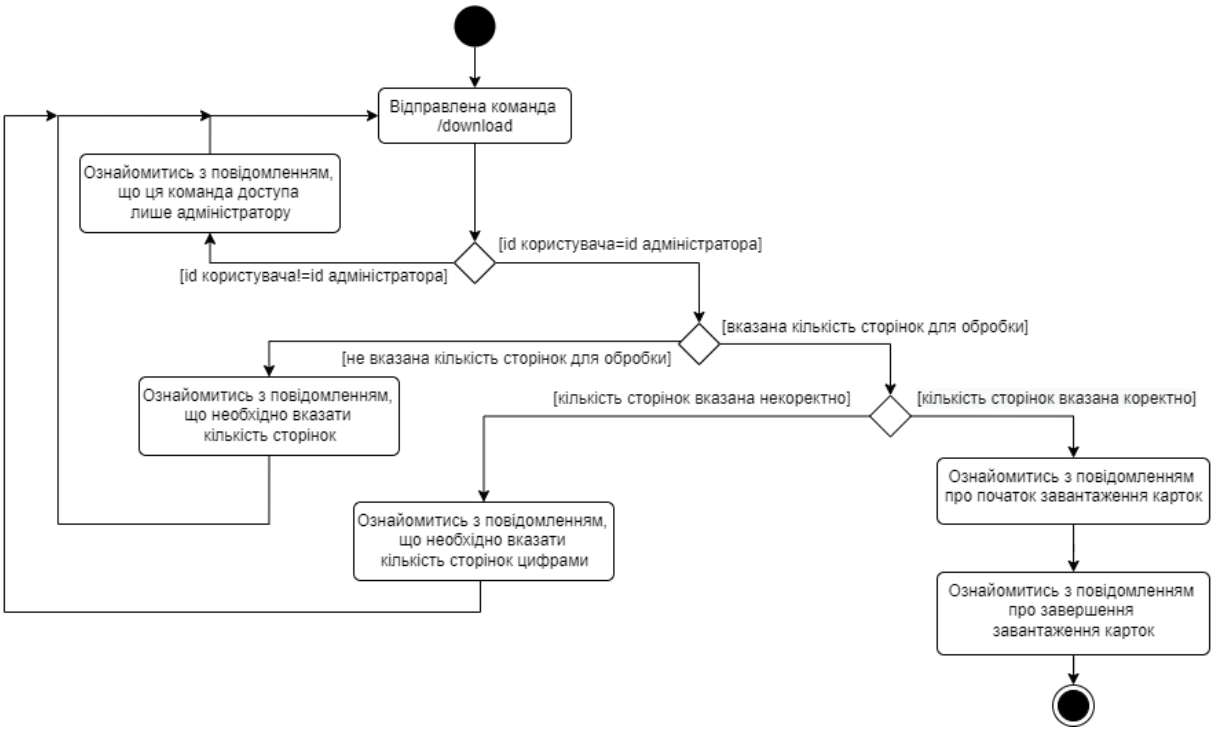

Рисунок 2.2 – Діаграма діяльності для актора «Адміністратор»

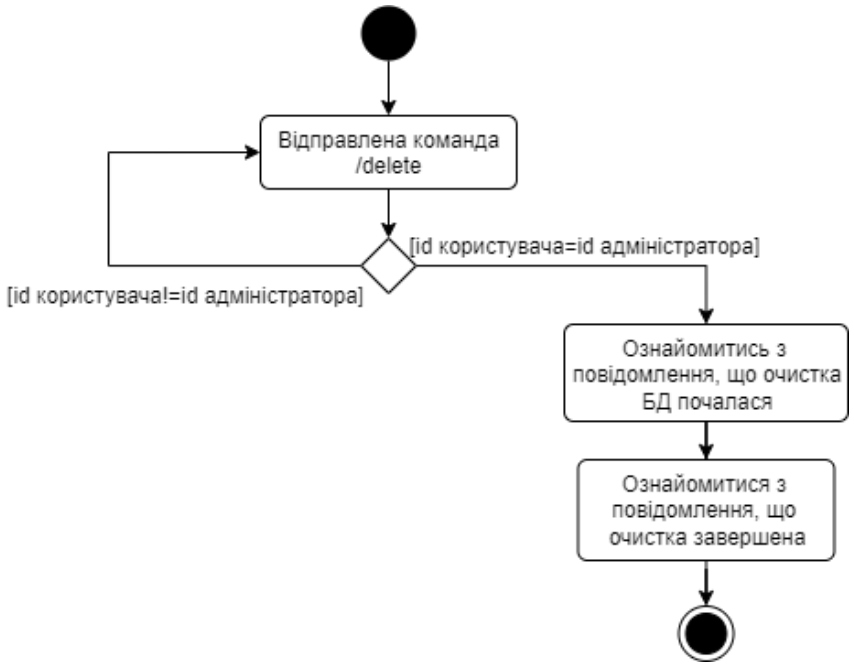

Рисунок 2.3 – Друга діаграма діяльності для актора «Адміністратор»

На рисунку 2.4 показано розроблену діаграму кооперації при завантаженні карток до бази.

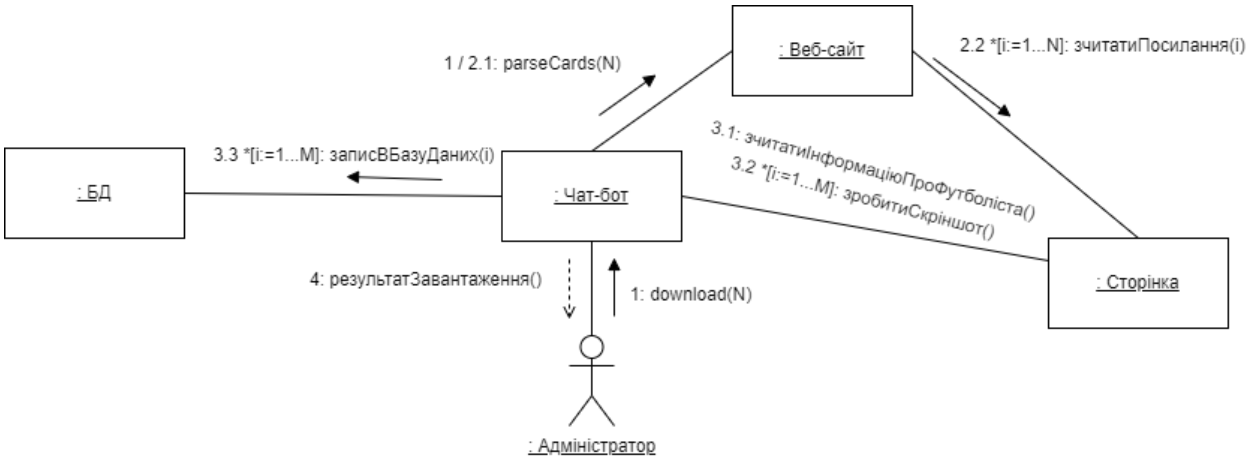

Рисунок 2.4 – Діаграма кооперації для актора «Адміністратор» – завантаження карток

Модуль для роботи з базою даних має містити такі функції:

- запуск бази даних та створення таблиць користувачів та футболістів, якщо такі не створені;
- додання нового користувача;
- додання інформації про футболіста;
- видалення запису про футболіста;
- вибрати ТОП-3 користувачів за кількістю набраних очок та конкретного користувача, якщо він не входить в ТОП-3;
- підрахунок кількості записів футболістів у базі;
- обрати інформацію про випадкового футболіста.

Для безпеки токен чат-боту та ID адміністратора винесено в окремий python-файл.

Безпосередньо ігровий процес має бути реалізований за допомогою машини станів з пам’яттю, оскільки необхідно запам’ятовувати кількість набраних очок, які підказки використано, варіанти відповідей та слідкувати за тим чи вгадав користувач, чи ні. Діаграма класів [14] для кінцевого автомату зображена на рисунку 2.5, а діаграма станів [21] системи в ігровому процесі на рисунку 2.6.

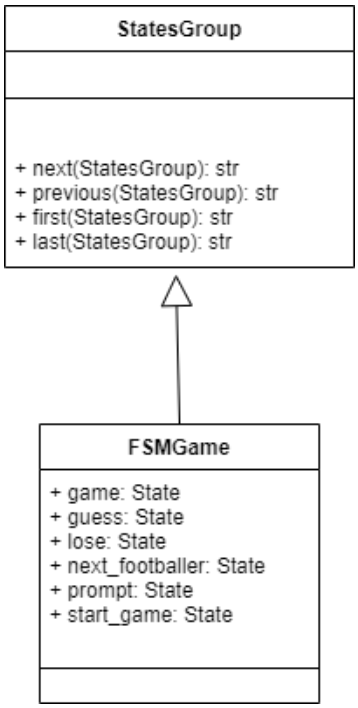

Рисунок 2.5 – Діаграма класів для кінцевого автомату

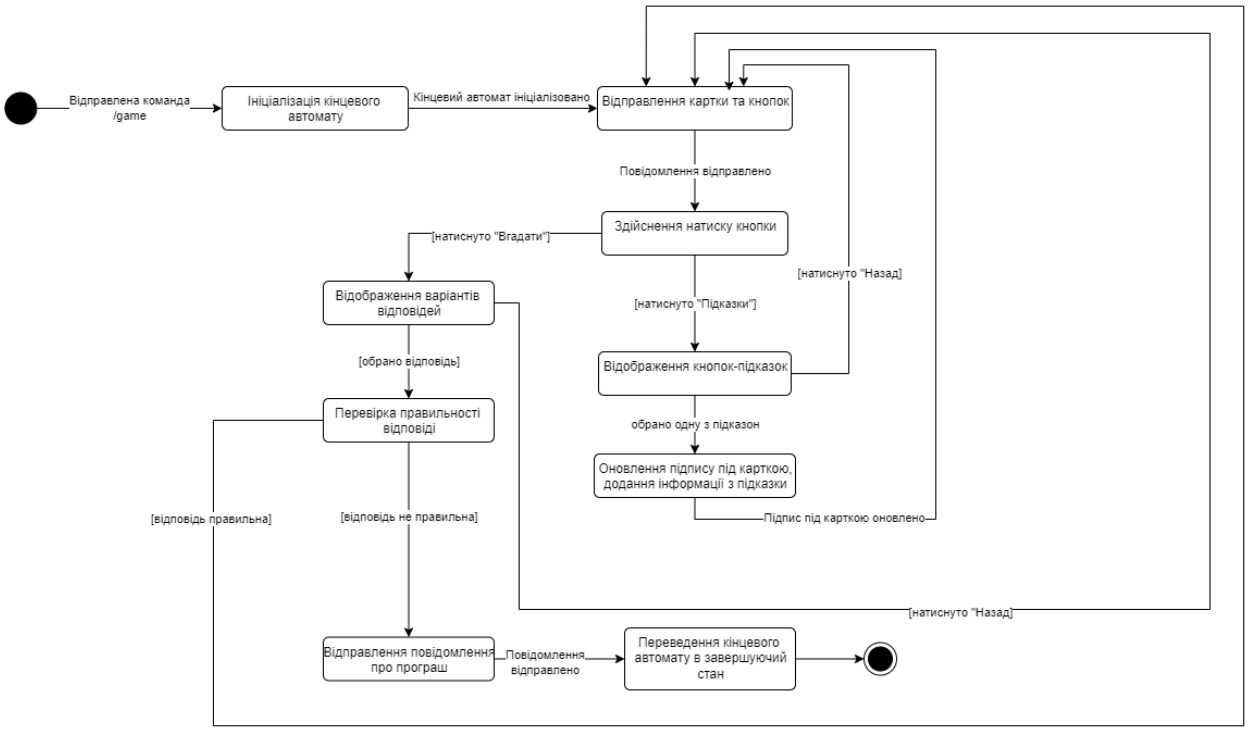

Рисунок 2.6 – Діаграма станів

Чат-бот надсилає зображення картки, на якому закрита інформація про футболіста. Користувач може дізнатися приховану інформацію за допомогою підказок, але використання кожної підказки віднімає один бал від

максимально можливих п’яти. Розроблена діаграма варіантів використання представлена на рисунку 2.7.

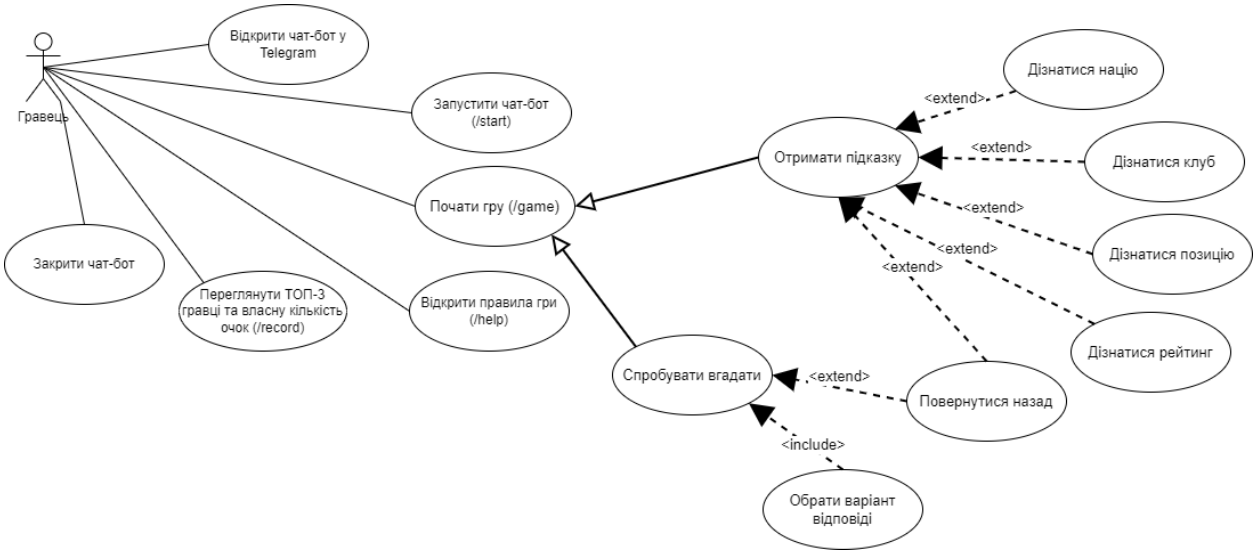

Рисунок 2.7 – Діаграма варіантів використання для актора «Гравець»

У разі, якщо користувач не вгадав, йому не нараховується жодного балу за поточного гравця, до бази даних заноситься загальна кількість балів, яку користувач заробив за гру. Для випадку, коли користувач використав підказку «клуб» та не вгадав розроблено діаграму послідовності (рис. 2.8) [22].

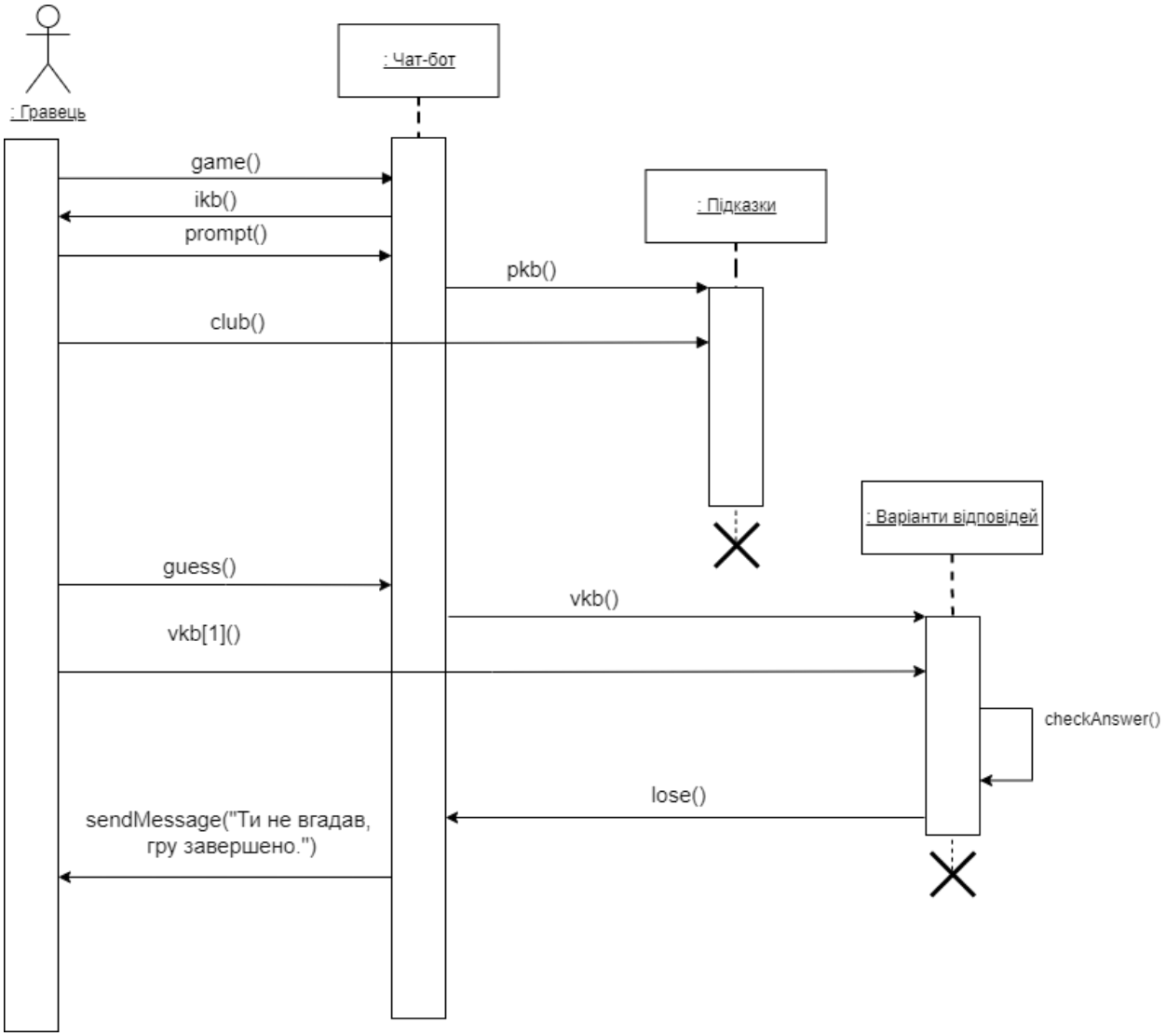

Рисунок 2.8 – Діаграма послідовностей для випадку невірної відповіді

Створення клавіатури та кнопок, які не відносяться до варіантів відповідей винесено в окремий модуль.

Кнопки з варіантами відповідей повинні створюватися динамічно, для того, щоб варіанти відповідей не повторювалися, і правильна відповідь не була на одному й тому ж місці. Тому, створення кнопок-варіантів відповідей потребує функції, яка генерує 4 унікальних, випадкових числа з заданого діапазону. Ця ж функція застосовується і для підбору футболістів для варіантів відповідей.

Для запуску роботи боту необхідно його розмістити на сервері або локальному комп’ютері. Щоб користувачі могли взаємодіяти з чат-ботом їм

необхідно лише встановити на смартфон чи будь-який інший девайс месенджер Telegram. Згідно з цими вимогами створено діаграму розміщення (рисунок 2.9).

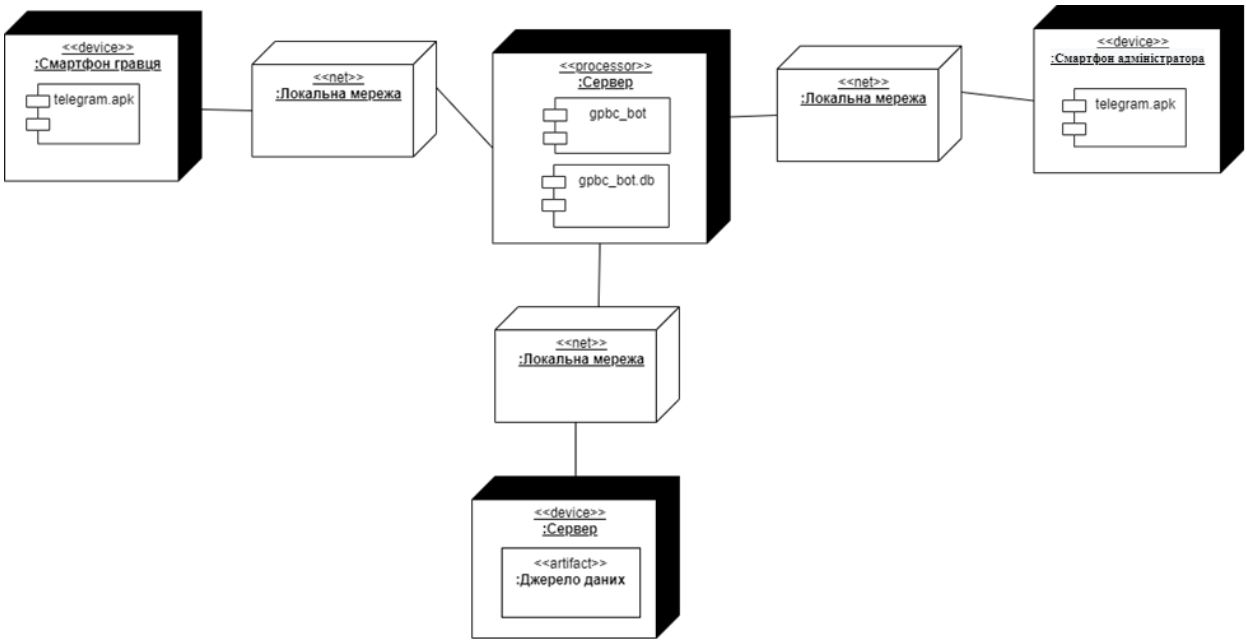

Рисунок 2.9 – Побудована діаграма розміщення

Побудовано діаграму пакетів для програмної системи (рис. 2.10).

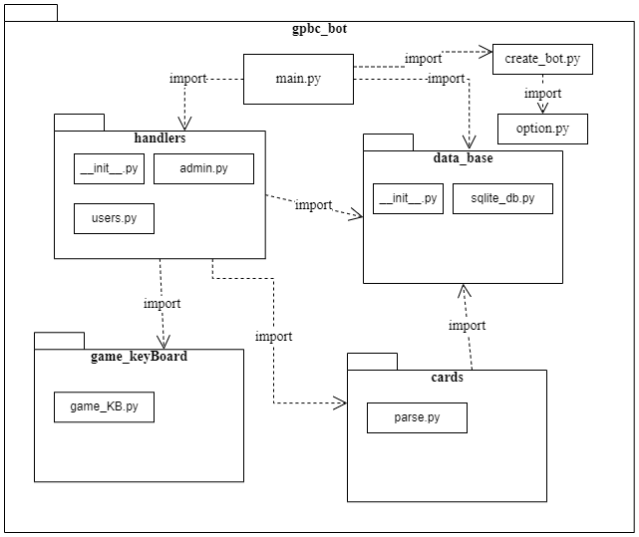

Рисунок 2.10 – Діаграма пакетів системи

2.1.1 Python-telegram-bot

Python-telegram-bot є чистою реалізацією Telegram Bot API для Python без додаткових надбудов. Ця особливість робить процес створення боту

простішим, оскільки можна подивитись офіційну документацію для Telegram Bot API.

Але у той самий час чиста реалізація викликає складнощі та збільшення кількості часу на розробку, оскільки деякі корисні надбудови, які є в інших бібліотеках, в python-telegram-bot відсутні.

2.1.2 Aiogram

Aiogram – зрозумілий, потужний і повністю асинхронний фреймворк для Telegram Bot API [4]. Переваги цього фреймворку:

- асинхронність;
- більш висока швидкодія та стійкість чат-боту;
- реалізована машина станів з пам’яттю;
- обробники помилок;
- обширна документація;
- фільтри для обробників та можливість створювати власні фільтри.

2.2 База даних

У стандартний пакет установки Python 3 входить модуль SQLite3.

SQLite – безсерверна, реляційна база даних. Для з’єднання з такою базою даних немає необхідності встановлювати сервер чи робити якісь налаштування.

Для більш зручного перегляду даних в таблицях додатково встановлено DB Browser (SQLite).

2.2.1 DB Browser (SQLite)

DB Browser – якісний графічний інструмент для створення, проєктування й редагування файлів бази даних SQLite [23].

Інтерфейс програми інтуїтивно зрозумілий. Програмний продукт дозволяє без знань мови SQL легко проводити необхідні маніпуляції з базою даних.

2.3 Веб-скрапінг

Для реалізації алгоритму веб-скрапінгу використано бібліотеки requests та BeautifulSoup4 [24].

Requests реалізовує HTTP-запити для взаємодії з сайтами.

BeautifulSoup4 слугує для вилучення даних з файлів HTML. Функціонал бібліотеки полегшує навігацію, пошук та редагування дерева HTML.

Картки на сайті-джерелі представлені з багатьох окремих елементів: кількох зображень та текстових даних. Для вирішення цієї проблеми було застосовано функції бібліотеки HTML2Image [25].

За допомогою цієї бібліотеки можна зробити скріншот сторінки потрібної сторінки певного розміру (рисунок 2.11).

Рисунок 2.11 – Приклад скріншоту, зробленого за допомогою HTML2Image

На отриманому зображенні багато зайвої інформації, тож скріншот слід додатково обробити. Необхідний функціонал реалізовано у бібліотеці PIL [26]. За допомогою цієї бібліотеки можна відкрити вже завантажене зображення, обрізати його та приховати інформацію-підказки про футболіста (рисунок 2.12).

Рисунок 2.12 – Зображення після обробки

2.4 Мова програмування та середовище розробки

Мовою програмування було обрано Python 3.10.4 – остання з доступних версій. У пакеті встановлення йде менеджер пакетів pip для завантаження бібліотек через консоль та модуль для роботи з базою даних SQLite3.

Середовищем для розробки було обрано PyCharm [27]. Ця IDE має два типи ліцензії – Professional Edition (платну) та Community Edition (безкоштовну), для розробки чат-боту була обрана остання. Інтерфейс програми представлений на рисунках 2.13-2.16.

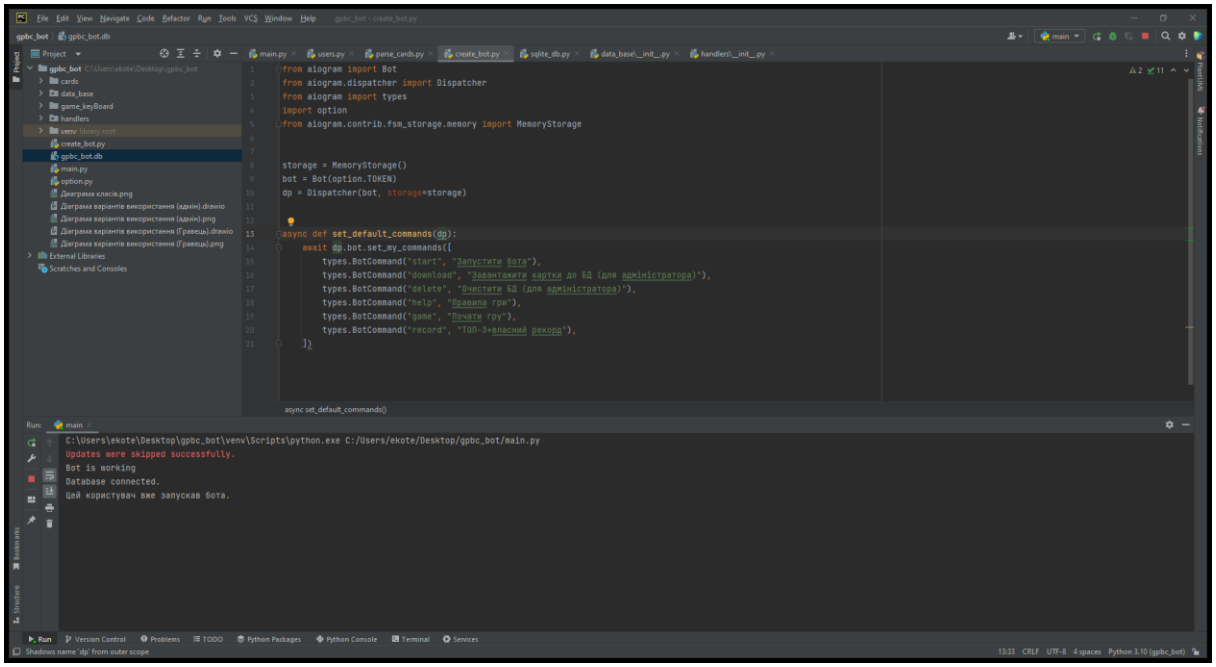

Рисунок 2.13 – Загальний інтерфейс PyCharm

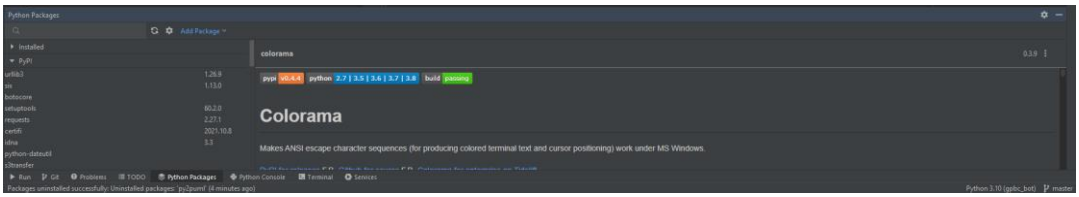

Рисунок 2.14 – Вбудований менеджер пакетів у PyCharm

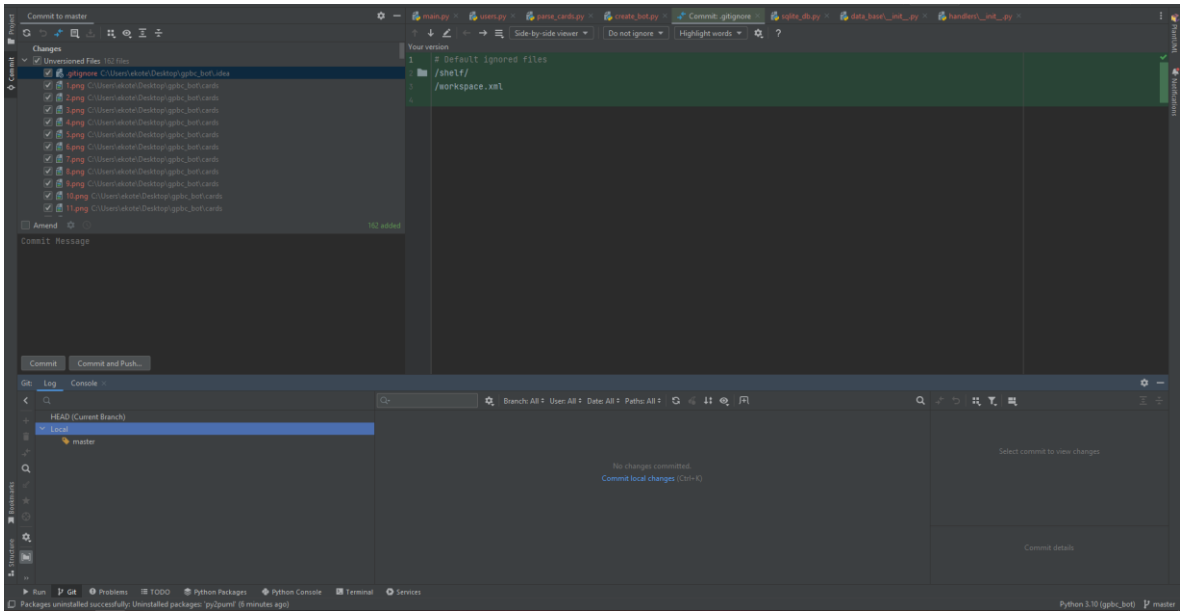

Рисунок 2.15 – Робота з системою контролю версій git

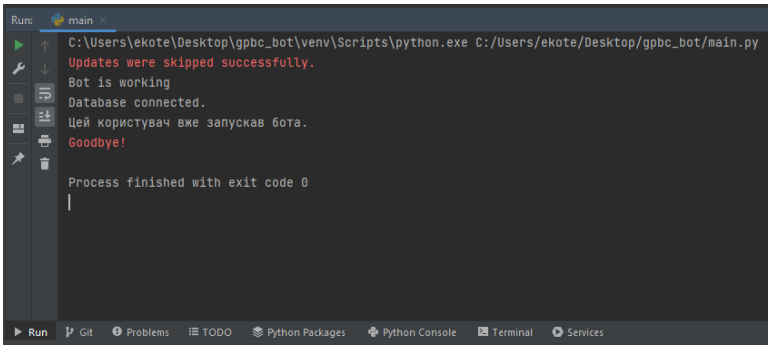

Рисунок 2.16 – Консоль PyCharm

PyCharm має ряд переваг перед конкурентами:

– автодоповнення коду;

– підсвітка помилок й швидке виправлення;

– автоматичний рефакторинг;

– інтерфейс для завантаження python-бібліотек;

– відображення файлової структури проєкту;

– зручна навігація по проєкту та коду;

– вбудований відлагоджувач;

– підтримка систем контролю версій git, mercurial та інші;

– можливість роботи у командному рядку PowerShell.

2.5 Інструмент для побудови UML-діаграм

Для побудови UML-діаграм існує багато інструментальних засобів. Найбільш відомими серед них є: Draw.io, Microsoft Visio, Gliffy, Rational Rose.

Для створення UML-діаграм та блок-схем, які розроблені при проєктуванні чат-боту, було обрано сервіс Draw.io [28]. Цей інструмент дозволяє створювати різні типи UML-діаграм, блок-схеми алгоритмів та, навіть, бізнес-процесів.

Draw.io володіє такими перевагами:

– безкоштовне розповсюдження;

– робота у web-версії або у додатку на комп’ютері чи смартфоні;

– можливість редагувати стиль елементів;

– вибір готових діаграм;

– широкий набір елементів;

– кросплатформеність;

– доступні популярні формати JPG, PNG, SVG для зберігання діаграм;

– можливість взаємодіяти з сервісами Google Drive, OneDrive, Dropbox, GitHub.

На рисунку 2.17 можна ознайомитися з наборами шаблонів діаграм та складових елементів діаграм.

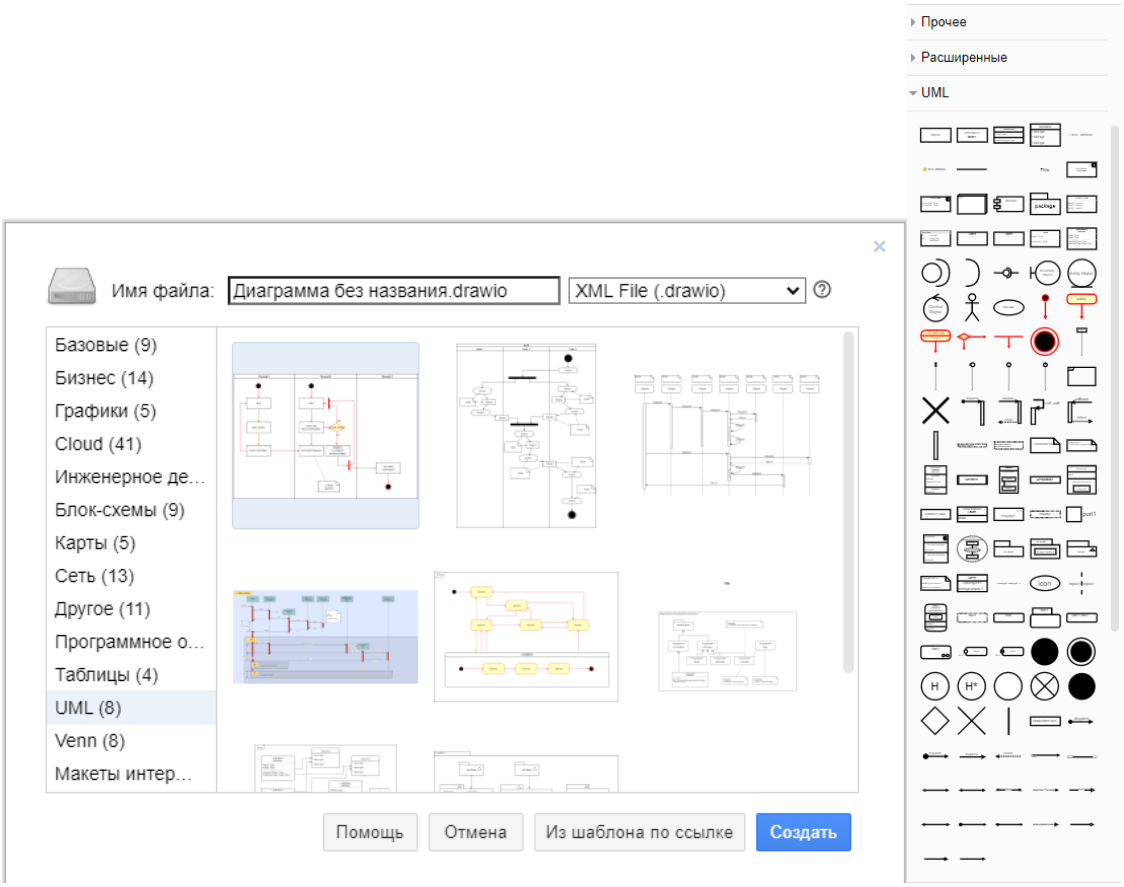

Рисунок 2.17 – Набір шаблонів діаграм та елементів у Draw.io

На рисунку 2.18 зображено загальний інтерфейс програми, справа знаходиться панель для редагування стилю елемента.

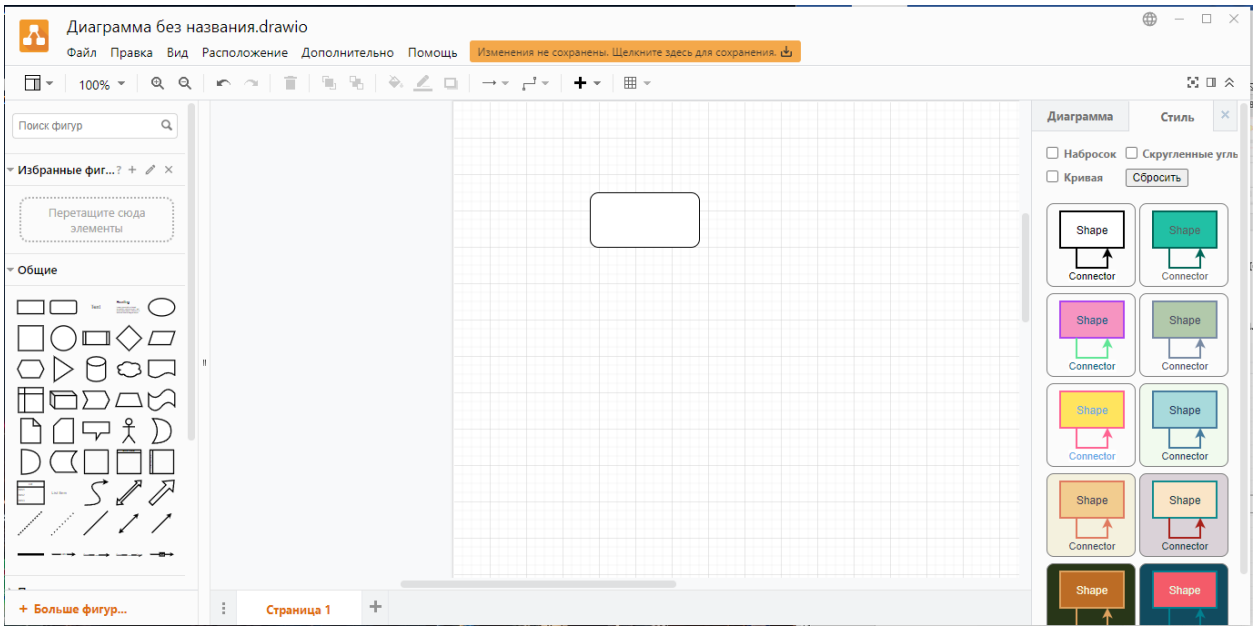

Рисунок 2.18 – Інтерфейс Draw.io

2.6 Висновки за розділом

У другому розділі була представлена архітектура чат-боту, описана взаємодія між модулями, процеси, які виконують модулі та команди. Було розглянуто методи вирішення поставлених задач.

Бібліотекою, для взаємодії з Telegram Bot API було обрано Aiogram, для парсингу використано бібліотеки requests та BeautifulSoup4, для роботи з графічними зображеннями – HTML2Image та PIL, робота з базою даних реалізовуватиметься за допомогою вбудованого модуля SQLite3.

Для написання програмного коду було обрано мову Python 3.10.4 та безкоштовну версію середовища розробки PyCharm. Для створення UML-діаграм та блок-схем було обрано програму Draw.io.

3 СТВОРЕННЯ ТА ФУНКЦІОНАЛЬНЕ НАВАНТАЖЕННЯ МОДУЛІВ ЧАТ-БОТУ

3.1 Створення нового боту у Telegram

Для реєстрації нового чат-боту у Telegram, необхідно знайти офіційний бот з іменем BotFather (рис. 3.1).

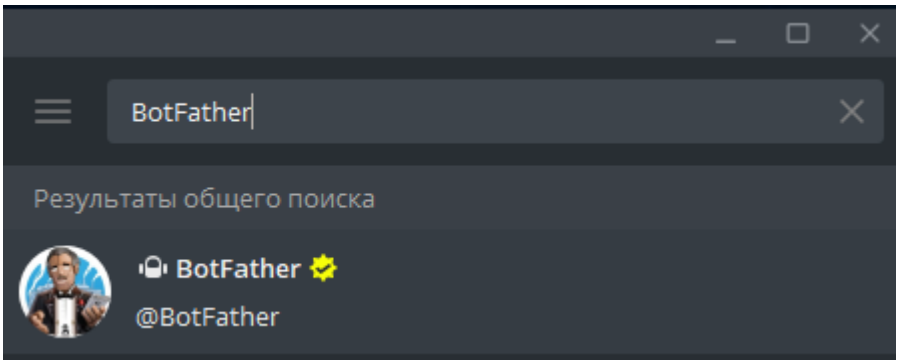

Рисунок 3.1 – Головний бот Telegram

Розпочавши діалог з цим ботом, слід ввести команду /newbot, після чого буде запропоновано обрати ім’я та username для нового боту.

Для боту було обрано ім’я – «Вгадай футболіста за карткою» та username – guessFootballerByCardBot.

Якщо обране ім’я та нікнейм вільні, BotFather відправить повідомлення, у якому буде посилання на бота та Token для управління ботом за допомогою API (рисунок 3.2).

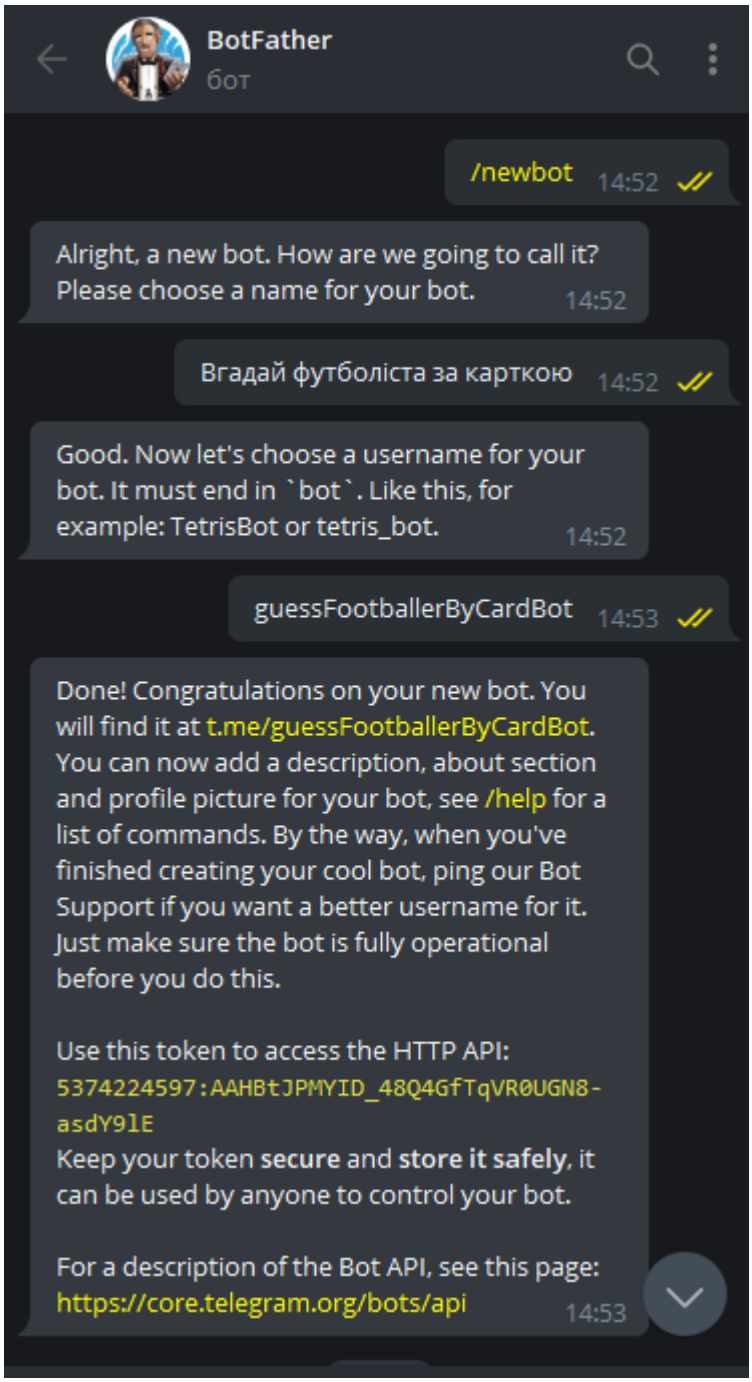

Рисунок 3.2 – Створення нового боту

Для персоналізації використано команду /mybots, після чого за допомогою inline-клавіатури обрано бота, дані про якого необхідно відредагувати. Таким чином було встановлено фото (рис. 3.3), опис та інформацію про бота (рис.3.4).

Рисунок 3.3 – Встановлено фото для боту

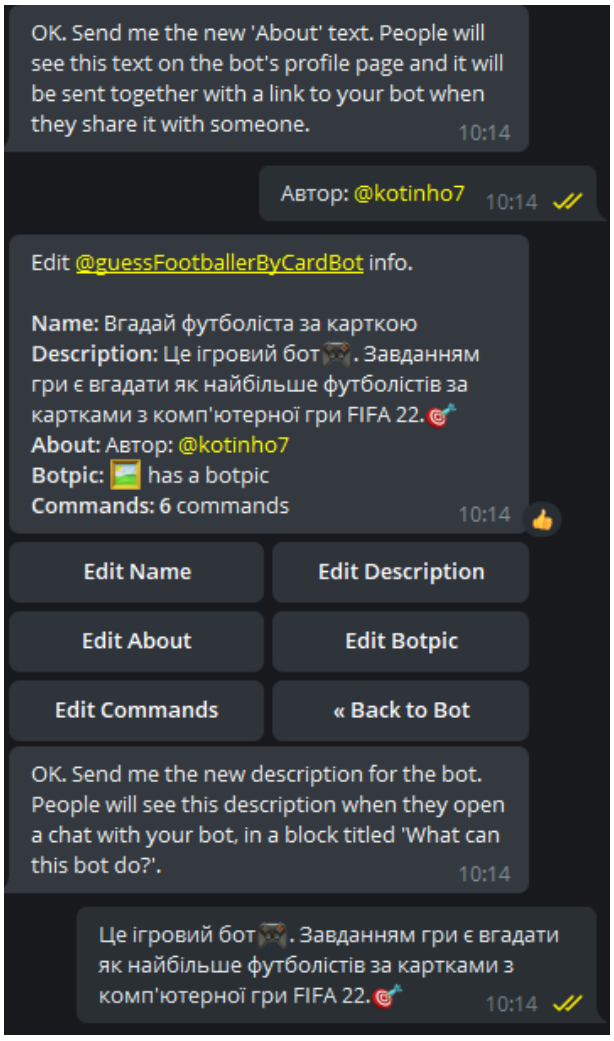

Рисунок 3.4 – Задано інформацію про бота та опис

3.2 Створення проєкту у PyCharm

Створено новий проєкт під назвою gpbc_bot, використовуючи віртуальне оточення Virtualenv та інтерпретатор Python3.10 (рис. 3.5).

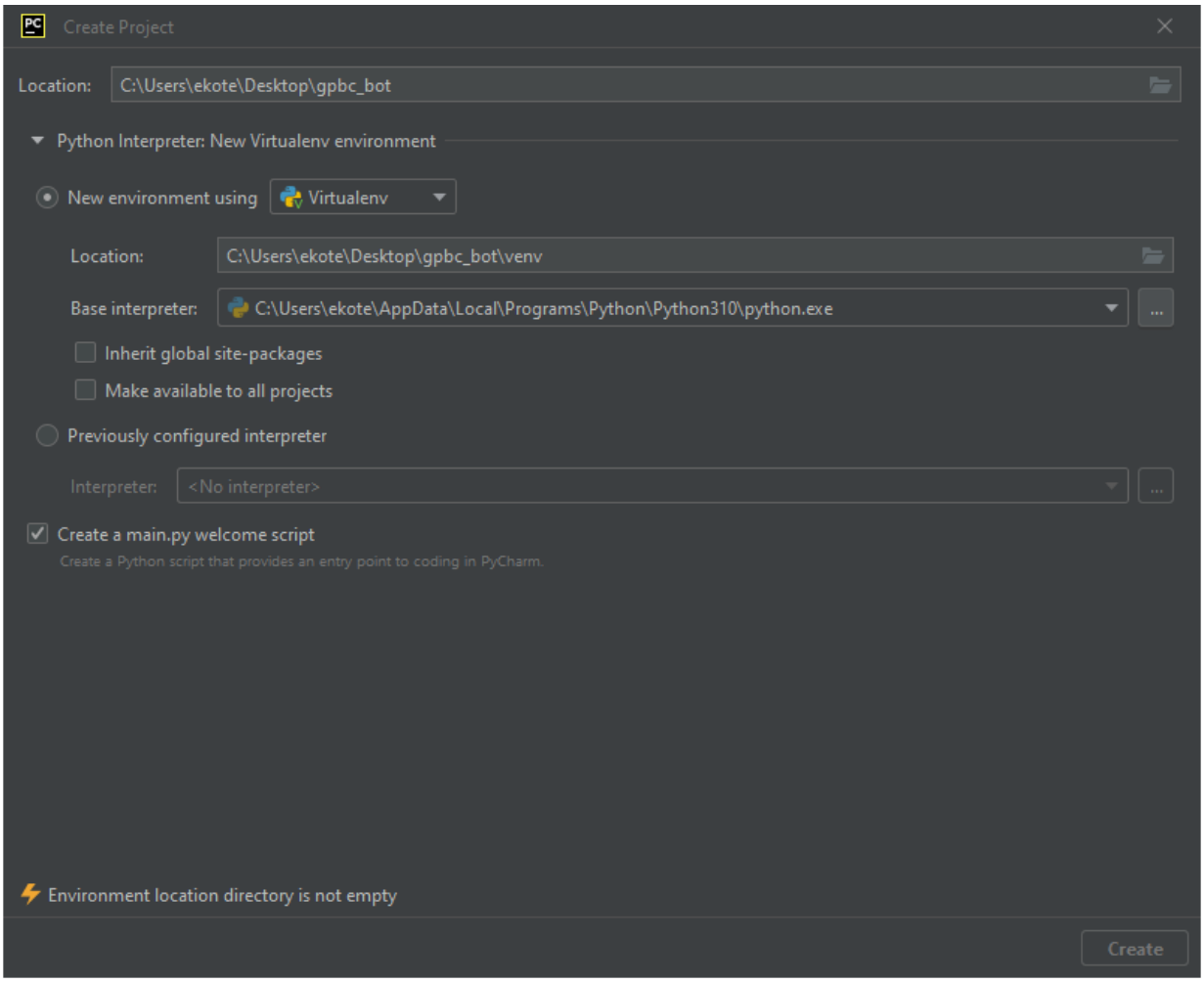

Рисунок 3.5 – Параметри створення нового проєкту

3.3 Модуль create_bot.py

Призначення цього модулю – створення екземпляру боту і диспетчера для керування ботом. Також у модулі створюється об’єкт, який дозволить зберігати дані в оперативній пам’яті, це необхідно, для запам’ятовування використаних підказок в ігровому процесі та нарахуванні балів за гру.

У цьому модулі реалізовано функцію, яка реєструє команди чат-боту. Було створено такі команди:

– /start – запустити бота;

– /help – правила гри;

– /game – почати гру;

– /record – переглянути ТОП-3 та власну кількість очок;

– /download – команда адміністратора для завантаження карток до БД;

– /delete – команда адміністратора для очищення таблиці з футболістами.

Екземпляр боту є об’єктом класу Bot, диспетчер – клас Dispatcher, пам’ять – MemoryStorage. Усі використані класи імпортовані з бібліотеки Aiogram.

При ініціалізації боту аргументом конструктора є токен для взаємодії з Telegram Bot API. Токен винесений в окремий модуль option.py у змінну типу str (додаток В.8).

При створенні об’єкту диспетчер, параметрами конструктору є екземпляр боту та об’єкт пам’яті.

Лістинг модуля міститься у додатку В.1.

3.4 Модуль main.py

Модуль розроблено для запуску боту, для цього виконується імпорт диспетчера.

Для запуску боту існує 2 режими: LongPolling та Webhook.

Метод LongPolling дозволяє запустити бот на локальній машині. У цьому режимі бот виконує запити на сервер Telegram про наявність повідомлень для нього. Цей метод добре підходить для процесу розробки і тестування боту, при великій кількості одночасних користувачів можлива нестабільна робота боту, може виникнути необхідність перезапустити його.

У режимі Webhook, бот завантажується на сервер, боту надається URL адреса. Коли виникає повідомлення для боту, сервер Telegram сам надсилає запит боту.

При виконанні роботи використано метод LongPolling, оскільки навантаження не суттєве.

З пакета aiogram.utils імпортовано модуль executor, який містить функцію start_polling для запуску бота в режимі LongPolling, аргументами в функцію передано диспетчер, функцію on_startup та прапорець ігнорування повідомлень, надісланих, коли бот не працював, встановлено в True.

Функція on_startup запускається одразу, як тільки сам бот починає працювати, за допомогою цієї функції виконується підключення до бази даних та викликається функція з модулю create_bot, яка встановлює команди для боту. Сформоване меню команд показано на рисунку 3.6.

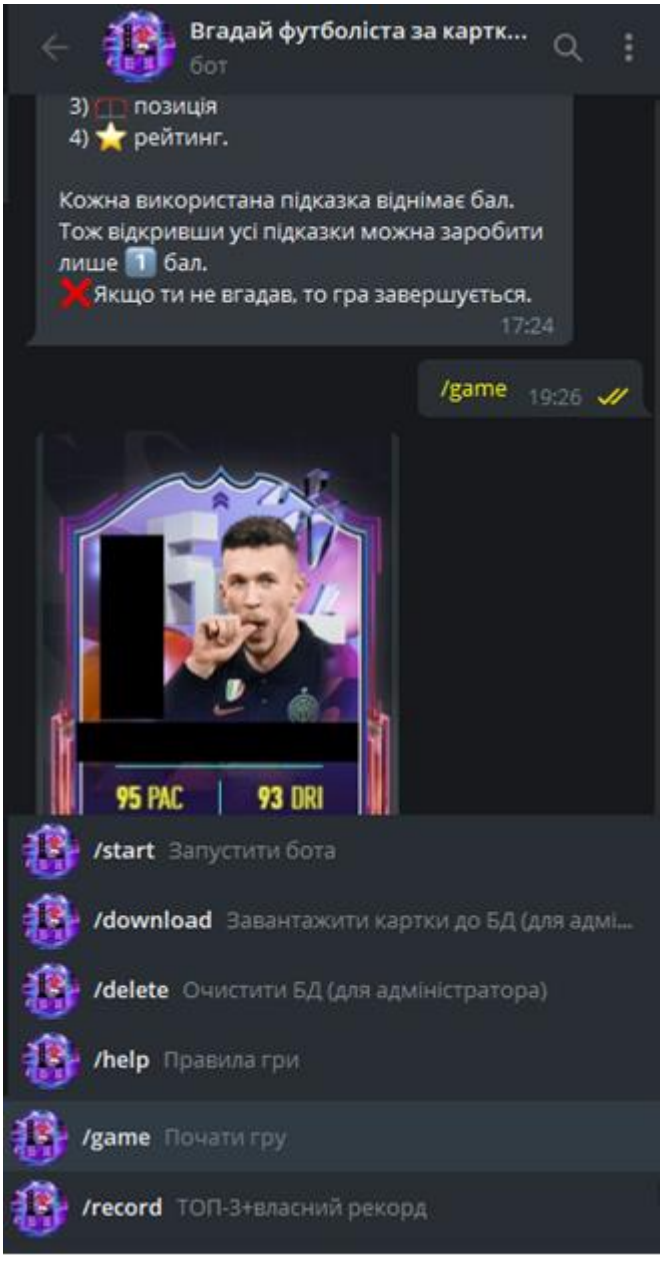

Рисунок 3.6 – Меню команд боту

Також реєструються обробники команд та кнопок, які написані в окремому пакеті handlers, першим викликається реєстратор обробників модуля users, після нього – admin.

Лістинг модуля міститься у додатку В.2.

3.5 Модуль sqlite_db.py

Даний модуль розроблено для роботи з базою даних, для коректної взаємодії з основним модулем програмної системи, цей модуль винесено в окремий пакет data_base, який ініціалізується у файлі __init__.py (додаток В.8).

База даних складається з двох таблиць – user_score та footballer. Поля таблиці, первинні ключі, типи даних та обмеження відображено на рисунку 3.7.

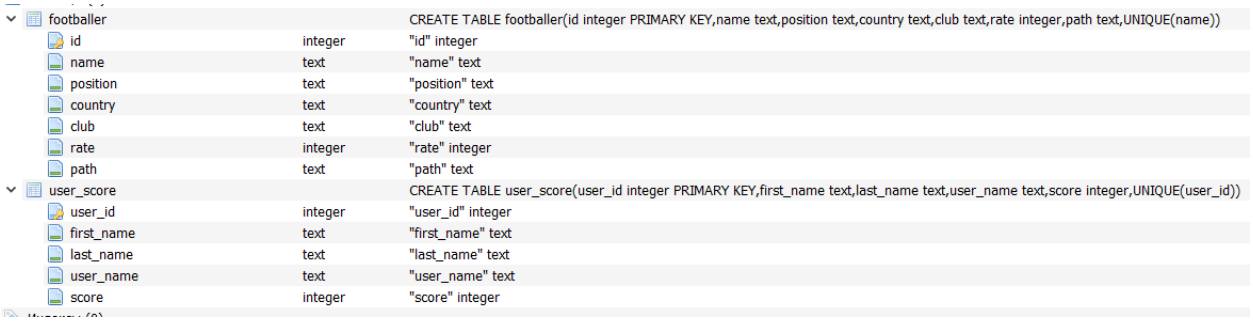

Рисунок 3.7 – Структура таблиць бази даних

У цьому модулі розроблено такі функції:

– sql_start – підключення до існуючої бази даних або створення, якщо її немає

– sql_add_user – функція вносить дані нового користувача у БД;

– sql_add_footballer – функція записує дані про футболіста;

– sql_del_footballer – очищення таблиці footballer;

– sql_record_user – переглянути ТОП-3 користувачів та власний рекорд;

– sql_count_footballer – повертає загальну кількість карток футболістів в базі даніх;

– sql_random_footballer – обирає футболіста за вказаним id;

– sql_update_record – функція записує нову кількість очок, якщо користувач покращив власний рекорд;

– sql_get_record – повертає кількість очок, набраних користувачем, для порівняння з поточною кількістю.

У функції sql_add_user такі аргументи: id користувача, ім’я, прізвище та username (якщо є) в Telegram.

Функція sql_add_footballer приймає в якості аргументів ім’я футболіста, позицію, країну, клуб, загальний рейтинг та порядковий номер.

Функція sql_record_user приймає один параметр – id користувача у Telegram.

Функція sql_random_footballer приймає цілочисельний параметр – випадково-згенероване число.

Аргументами функції sql_update_record є цілочисельний параметр кількості набраних очок та id користувача у Telegram.

Функція sql_get_record приймає один аргумент – id користувача.

Програмний код модуля наведено у додатку В.4.

3.6 Пакет handlers

Пакет ініціалізовано за допомогою файлу __init__.py (додаток В.8). У цьому пакеті міститься два модулі:

– admin.py – для обробки команд delete та download, і обробки текстових повідомлень;

– users.py – модуль обробки решти команд та кнопок у ігровому процесі.

3.6.1 Модуль admin.py

Модуль складається з функцій-обробників та функції, яка реєструє ці обробники.

У спеціальну змінну зчитується id адміністратора боту, яке записано у конфігураційному файлі option.py. Id необхідне для того, щоб запускати команди download та delete міг лише адміністратор.

Обробник команди download призначений для завантаження інформації про футболістів у БД та підготовки зображень карток до гри. На початку, виконується перевірка відповідності id користувача з id адміністратора. Після успішної перевірки функція намагається виділити з повідомлення число сторінок, які необхідно обробити. Таким чином, адміністратору необхідно одним повідомленням через пробіл вказати поруч з командою число сторінок для обробки. Якщо все вказано вірно, викликається функція веб-скрапінгу з відповідного модуля parse_cards.py.

При некоректному запуску команди передбачені відповідні повідомлення для користувача.

Обробник команди delete виконує очистку таблиці footballer та видаляє картки з локальної машини, на якій запущено бот. Аналогічно обробнику download, спочатку виконується перевірка id на відповідність. У випадку успішної перевірки викликається функція delete з модуля parse_cards.py.

Також у цьому модулі написано обробник текстових повідомлень, який інформує користувача, про те, що взаємодіяти з ботом необхідно за допомогою команд та кнопок, а повідомлення користувача видаляє. Цей обробник стоїть на останньому місці

Останньою функцію модуля є функція, яка реєструє вищевказані обробники.

Лістинг представлено у додатку В.5.

3.6.2 Модуль users.py

У цьому модулі реалізовано ігровий процес. Для виконання цієї задачі використано машину станів. Створено клас FSMGame, який успадковується від класу StatesGroup. Діаграма класів зображена на рисунку 3.8.

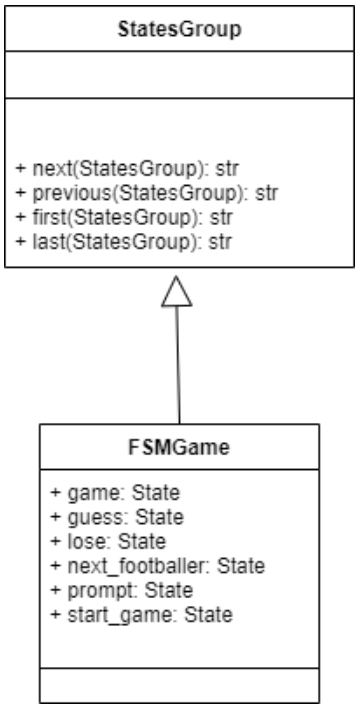

Рисунок 3.8 – Діаграма класів

Для реалізації ігрового процесу використано патерн проєктування «Стан».

Створений клас має такі поля типу State, які є станами кінцевого автомату:

– start_game – початок ігрового процесу;

– prompt – стан підказок;

– game – ігровий процес;

– guess – стан вгадування;

– next_footballer – наступний футболіст;

– lose – завершення ігрового процесу, користувач не вгадав.

Також цей модуль містить обробники команд start, help та record, які відправляють відповідні повідомлення користувачеві.

При спрацюванні обробника команди start дані про користувача заносяться до бази даних, якщо користувач раніше не запускав бота. У відповідь на кожну з команд користувач отримує відповідне повідомлення.

Також написано оброблювач повідомлень, які користувач спробуватиме відправляти боту під час ігрового процесу, такі повідомлення просто видалятимуться. Оброблювач розташовано у кінці перед функцією реєстратором обробників, це необхідно для того, щоб в нього потрапляли ті повідомлення, для яких не створено обробник.

Обробник команди game приймає у якості параметра стан кінцевого автомата None, тобто відбувся запуск машини станів. Одразу після запуску обробника встановлюється стан start_game. Для того, щоб надіслати якусь картку, необхідно за порядковим номером витягти дані про футболіста з бази даних.

Для того, щоб картки не йшли у порядку зростання порядкового номеру, реалізовано функцію, яка генерує 4 випадкові числа з двома аргументами – лівою і правою межею генерування. Чотири числа необхідні, щоб одразу отримати прізвища та імена футболістів для варіантів відповідей. Лівою межею є 1, а правою – кількість записів у таблиці footballer, яка отримується викликом спеціальної функції з модуля взаємодії з базою даних.

Згодом, у словник записується дані про правильну відповідь, а саме прізвище та ім’я, позиція, національність, клуб, рейтинг та шлях, де збережена картка футболіста, також записуються ім’я та прізвища ще 3 футболістів, які будуть серед варіантів відповідей. У словник також заноситься ключ score із значенням 0, та прапорці використаних підказок, де 0 – не використана, 1 – використана.

По завершенню роботи зі словником, бот надсилає користувачу повідомлення з зображенням картки та inline клавіатуру, яка згенерована у модулі game_KB.py. Встановлюється стан game.

Обробник кнопки «Підказки» редагує повідомлення боту, замінюючи клавіатуру на іншу та встановлює новий стан prompt. Клавіатура з підказками також генерується у модулі game_KB.py.

Далі йдуть 4 обробники для кожного виду підказок. Обравши будь-яку, система редагує підпис під надісланим зображенням, даючи текст підказки, і

замінює клавіатуру на основну. Прапорець використання змінює значення на 1. Кожна наступна підказка пишеться у підпису з нового рядка. При спробі повторного використання підказки, користувач отримає спливаюче сповіщення про те, що підказка вже була використана. Після завершення роботи обробників кнопок-підказок, стан машини змінюється на game.

Обробник кнопки «Вгадати» також використовує функцію генерування чотирьох випадкових чисел у заданому діапазоні, це необхідно для того, щоб варіанти відповідей змінювалися місцями. Далі згенерована клавіатура замінює основну клавіатуру. Встановлюється новий стан – guess.

У клавіатурах «Підказки» і «Вгадати» існує кнопка «Назад» для повернення до основної клавіатури, обробник кнопки змінює стан на game.

Обробник правильної відповіді редагує підпис під зображенням картки. Функція підраховує кількість використаних підказок, і визначає, скільки балів користувач отримав за вгаданого футболіста, додаючи до загальної кількості зароблених очок. Далі, встановлюється стан next_footballer, генеруються нові варіанти відповідей, прапорці підказок встановлюються у значення 0, у словник заносяться нові дані для підказок. Машина станів переходить у стан game.

Для неправильних варіантів відповідей написано один обробник. У спеціальну змінну зі словника передається кількість набраних користувачем очок. Видаляється клавіатура з-під зображення картки. Бот надсилає повідомлення про програш та кількість набраних очок. Машина переходить у стан lose.

Якщо набрана кількість очок більш, ніж дані, з бази даних, то заноситься нове значення, після чого машина станів завершує свою роботу.

Лістинг модуля у додатку В.6.

3.6.3 Модуль game_KB.py

Цей модуль створено в каталозі game_keyBoard. У модулі створено inline кнопки та дві клавіатури.

Inline клавіатура розташовується не під полем вводу повідомлення, а безпосередньо під відправленим повідомленням. Клавіатури, які створюються цим модулем зображені на рисунках 3.9-3.10.

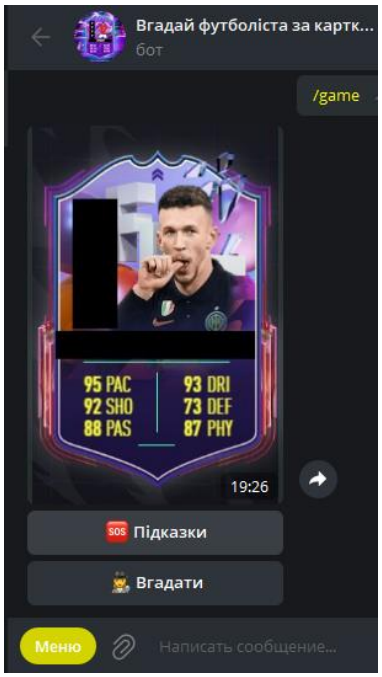

Рисунок 3.9 – Основна клавіатура

Рисунок 3.10 – Клавіатура з підказками

Клавіатура це об’єкт класу InlineKeyboardMarkup, при її створенні, параметром конструктора є ширина рядка, тобто кількість кнопок, які вміщаються в один рядок клавіатури.

Кнопка – екземпляр класу InlineButtonMarkup, конструктор цього класу приймає 2 параметри – текст, який відображається на кнопці, та дані, які повертає кнопка при натисканні на неї.

Щоб додати кнопку до клавіатури, використано метод add класу InlineKeyboardMarkup.

Лістинг надано у додатку В.7.

3.7 Модуль parse_cards.py

У цьому модулі реалізовано алгоритм веб-скрапінгу та функція очистки таблиці footballer і карток з машини, на якій запущено роботу боту.

Функція очистки записує у цілочисельну змінну кількість записів у таблиці footballer, викликаючи відповідну функцію з модуля sqlite_db, після чого з цього ж модуля викликається функція видалення записів з таблиці. Наступним кроком змінюється директорія на каталог cards, і в циклі видаляються зображення карток футболістів.

Функція веб-скрапінгу приймає параметром кількість сторінок для обробки. У циклі виконуються запити методом get до сайту-джерела. На кожній сторінці інформація про футболістів міститься у рядках таблиці, тому алгоритм за тегами та класами цих тегів у масив заносить посилання на сторінки футболістів. Цей процес виконується у циклі while з умовою, доки кількість оброблених сторінок не прирівняється до числа-параметра.

Наступний крок – цикл проходить по масиву посилань, виконуючи запит методом get отримує інформацію про футболістів, заносячи отримані дані у таблицю бази даних, спеціальною функцією з модулю sqlite_db. У цьому ж циклі необхідно сформувати зображення картки, для цього використано бібліотеку html2image і метод screenshot_url, який робить скріншот сторінки за посиланням необхідного розміру, і зберігає зображення на комп’ютер.

Отримане зображення містить зайву інформацію, тож його необхідно обрізати та приховати інформацію, яка використана для підказок. Цю задачу виконано за допомогою бібліотеки PIL. Створено екземпляр зображення, методом Image.open відкривається скрішнот, метод crop обрізає зображення. Далі створено копію отриманого зображення, щоб можна було малювати, використано метод ImageDraw.ImageDraw, параметром у яку передано обрізане зображення. Далі методом rectangle намальовано два прямокутники, і методом save збережено зображення у каталог cards. Проміжне зображення видалене методом os.remove.

Для розробленого алгоритму веб-скрапінгу побудовано блок-схему (рис. 3.11-3.12).

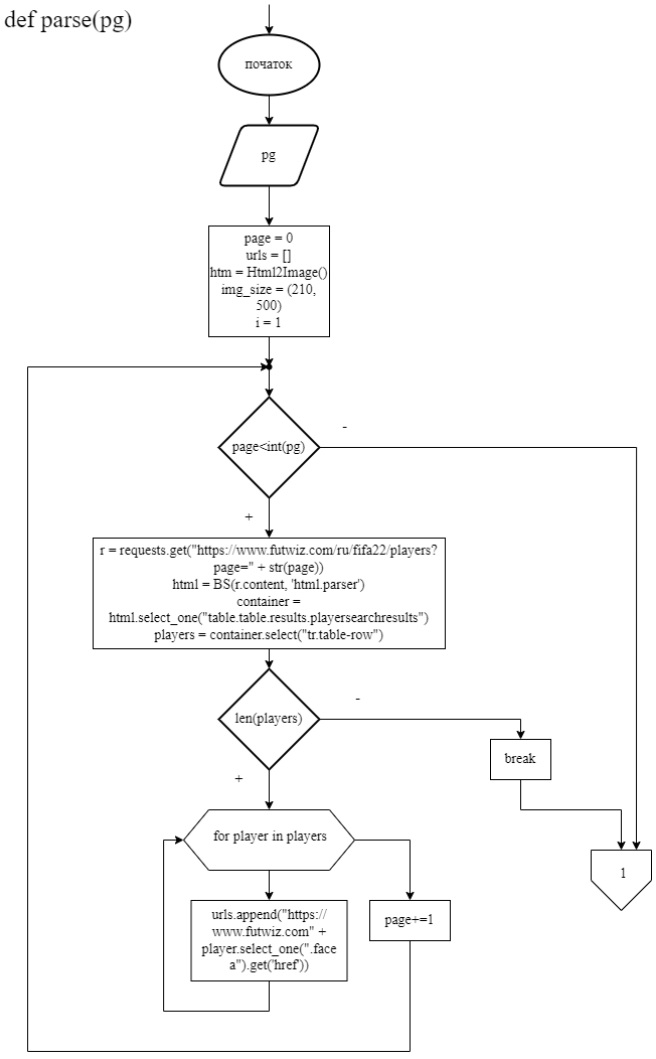

Рисунок 3.11 – Блок-схема алгоритму веб-скрапінгу

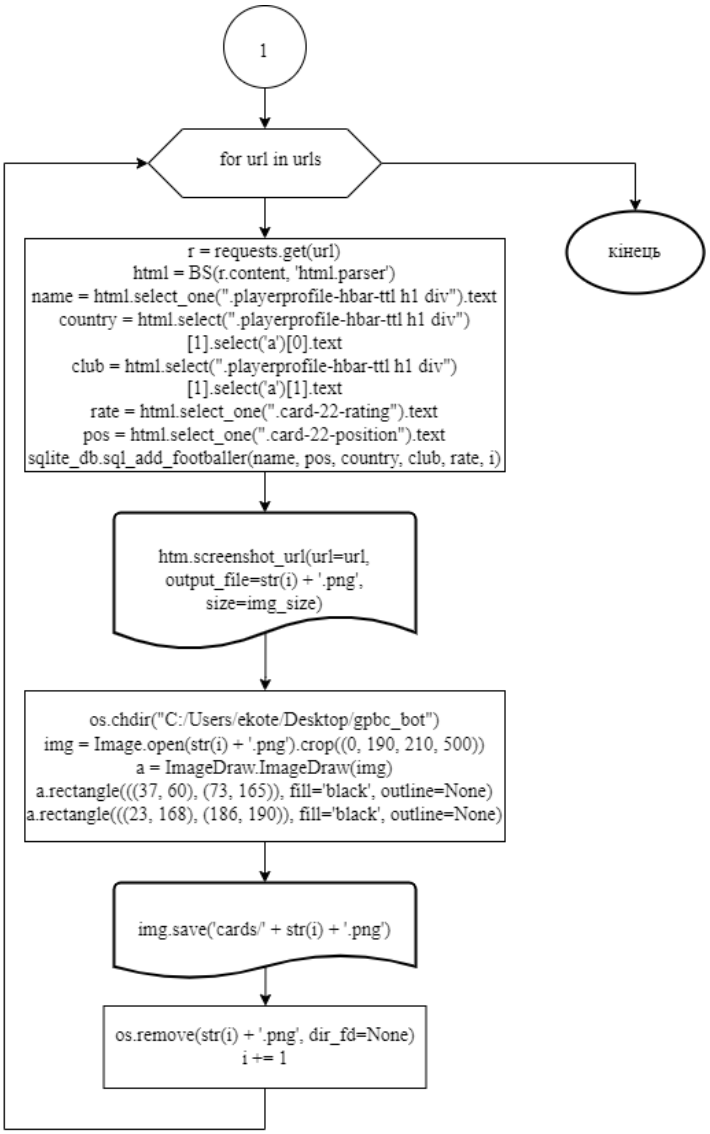

Рисунок 3.12 – Продовження

Лістинг модуля у додатку В.3.

3.8 Висновки за розділом

У цьому розділі описано реалізацію модулів програмної системи, наведено структуру створеної бази даних.

Також наведено діаграму класів, які використані, приведено блок-схему алгоритму веб-скрапінгу, продемонстровано приклади inline клавіатури.

Для реалізації ігрового процесу використано патерн проєктування «Стан».

4 ПРОГРАМНА РЕАЛІЗАЦІЯ РОБОТИ ЧАТ-БОТУ І ЗАСОБИ БЕЗПЕКИ

4.1 Програмна реалізація роботи системи

При запуску команди start, як і при першому запуску, бот надсилає короткий опис чат-боту та доступні команди (рис. 4.1).

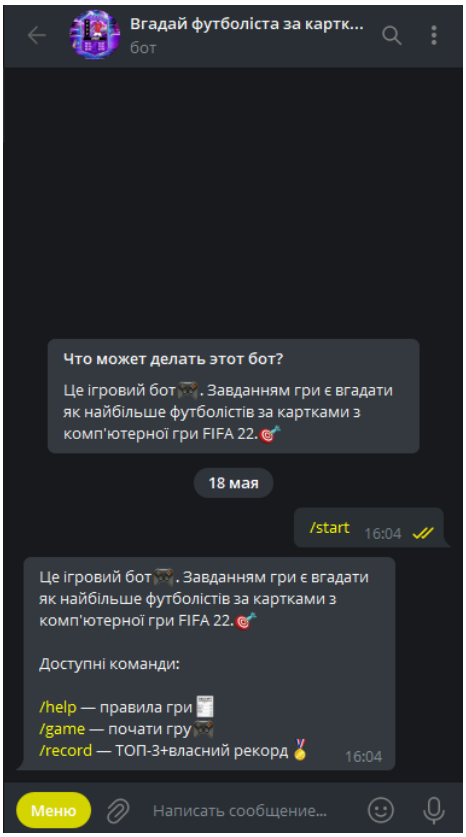

Рисунок 4.1 – Перший запуск чат-боту

При першому запуску дані про користувача вносяться до бази даних (рис. 4.2).

Таблиця: user_score

	user_id	first_name	last_name	user_name	score
	Фильтр	Фильтр	Фильтр	Фильтр	Фил...
1	302989396	Богдан	Передрий	shadowplayyy	0
2	350113681	necromancer		necromancerkrya	0

Рисунок 4.2 – Дані про користувачів, які вперше запустили бот

Запустивши команду help користувач отримує від чат-боту повідомлення, у якому описані правила гри (рис. 4.3).

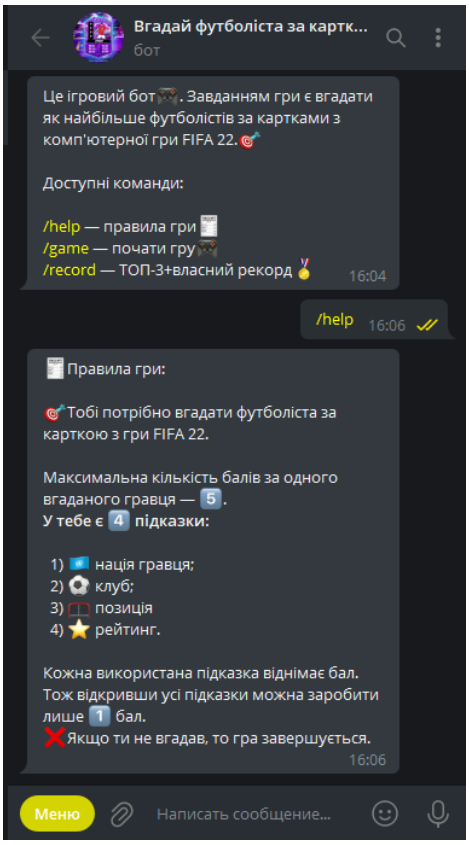

Рисунок 4.3 – Результат роботи команди help

За допомогою команди record можна переглянути топ-3 гравців та власний рекорд (рис. 4.4).

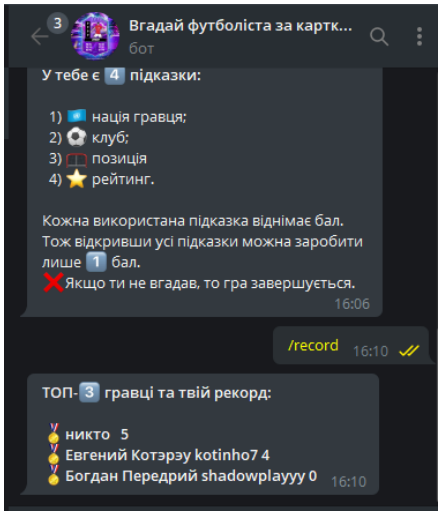

Рисунок 4.4 – Результат роботи команди record

Випадок, коли команду delete намагається запустити не адміністратор, продемонстровано на рисунку 4.5. Бот повідомляє, що ця команда доступна лише адміністратору, аналогічно з командою download.

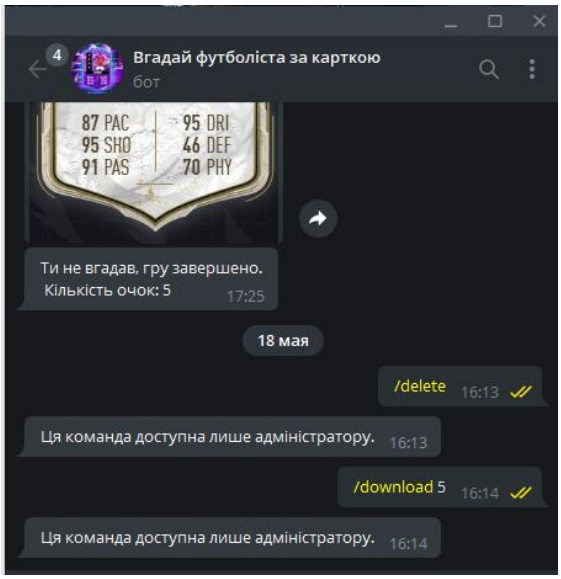

Рисунок 4.5 – Спроба запустити команди delete та download не адміністратором

На рисунку 4.6 продемонстрована некоректна спроба запустити команду download, це саме повідомлення буде надіслано, якщо замість числа будуть символи чи літери.

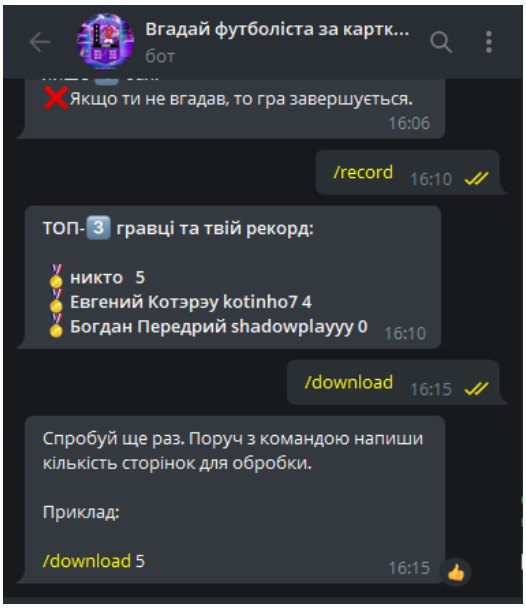

Рисунок 4.6 – Некоректна спроба запустити команду download

Приклад коректного запуску і відпрацювання команди download зображено на рисунку 4.7. Спочатку бот інформує про початок завантаження карток, а після завершення відповідно про кінець відпрацювання парсера.

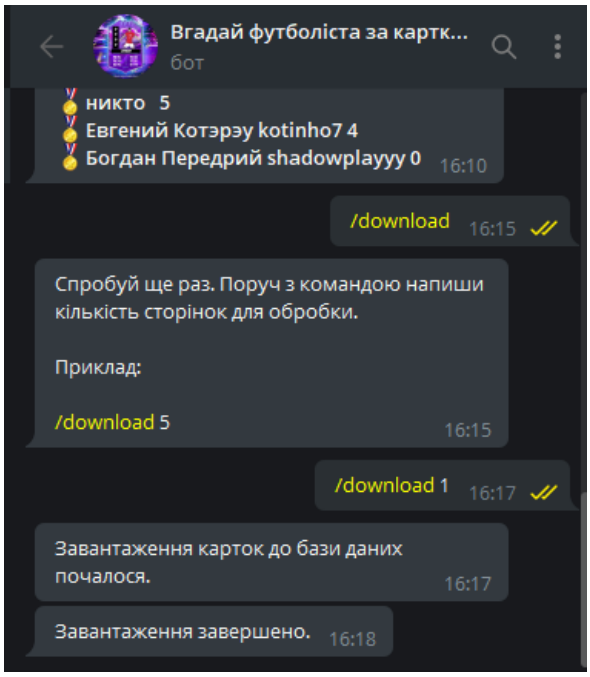

Рисунок 4.7 – Коректний запуск команди download

Результат роботи парсера показано на рисунку 4.8, де показано завантажені зображення краток та таблицю footballer.

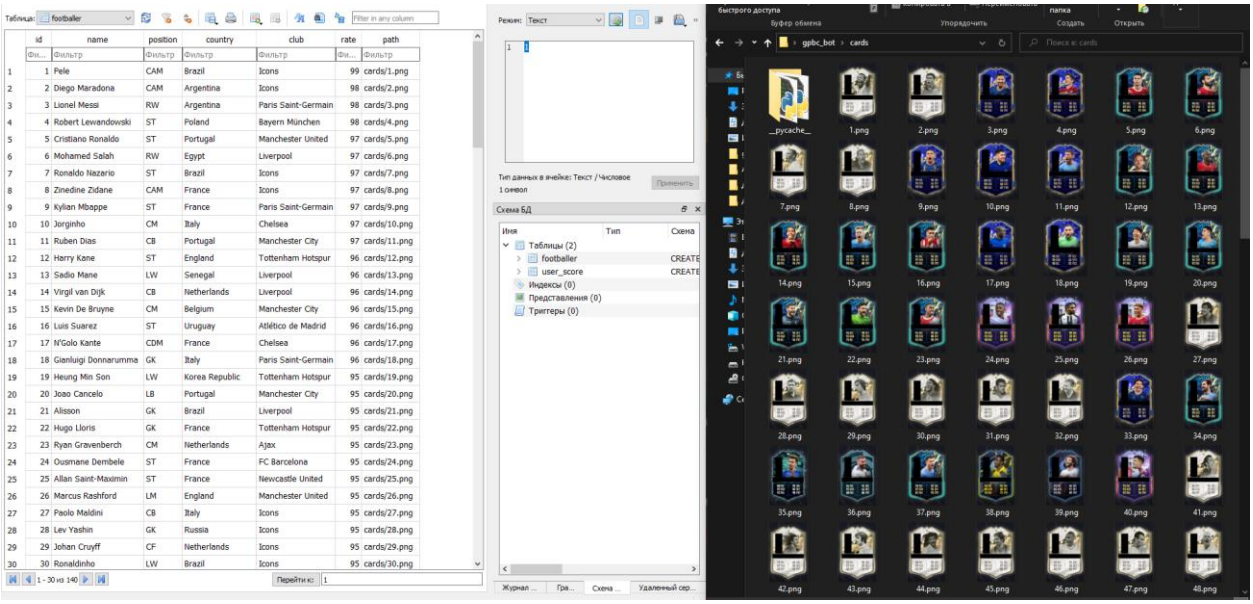

Рисунок 4.8 – Результат роботи алгоритму веб-скрапінгу

Відповідні повідомлення адміністратор отримує й про початок та завершення роботи команди delete, результат роботи команди на рисунку 4.10.

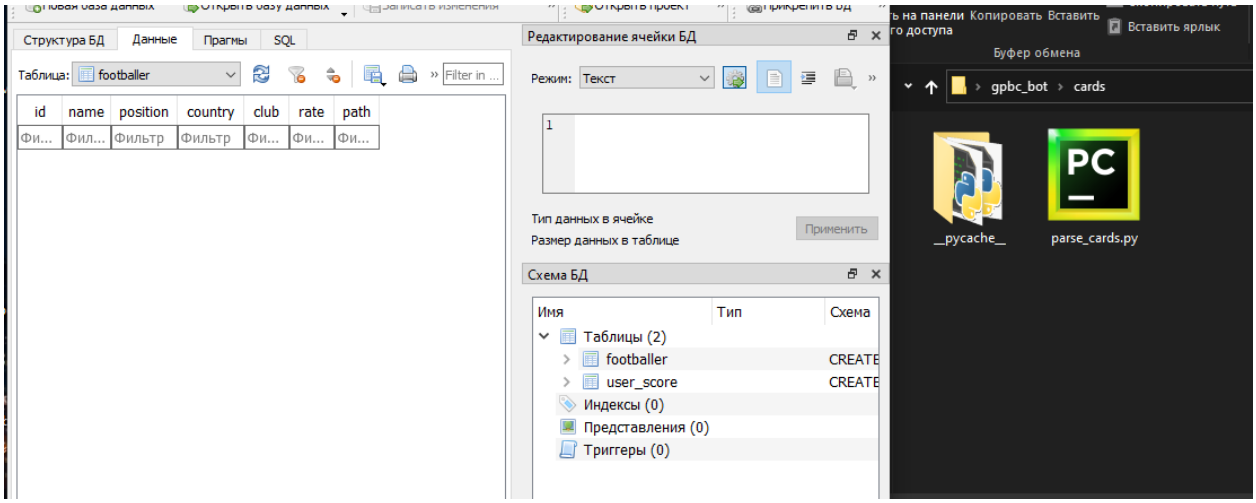

Рисунок 4.10 – Результат роботи команди delete

Якщо чат-боту відправити повідомлення, він видалить його і проінформує, що взаємодіяти з ним слід через команди та кнопки (рис. 4.11).

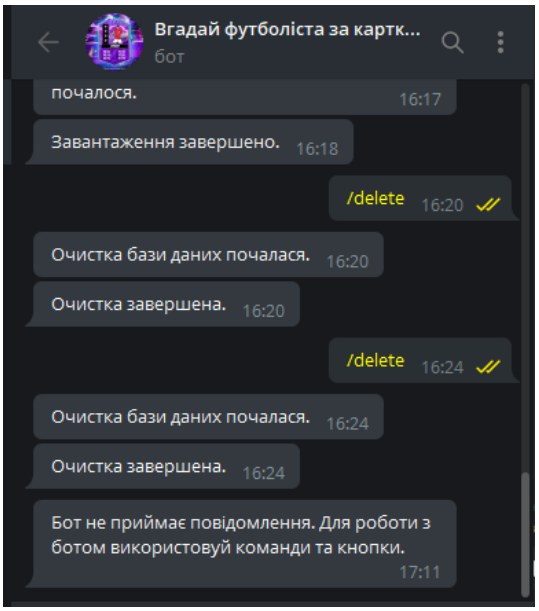

Рисунок 4.11 – Обробка повідомлень користувача

Командою game буде запущено машину станів, чат-бот відправить перше зображення футболіста (рис. 4.12).

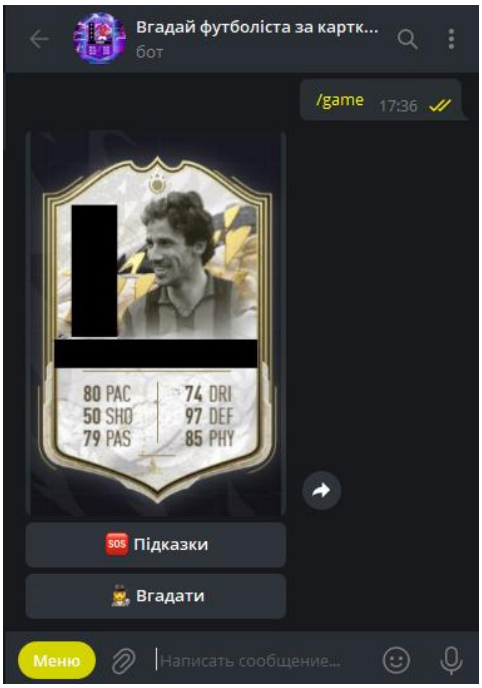

Рисунок 4.12 – Запущено команду game

Коли користувач натисне кнопку «Підказки» поточна клавіатура буде замінена на клавіатуру з підказками (рис. 4.13).

Рисунок 4.13 – Користувач натиснув кнопку «Підказки»

Натиснувши кнопку «Назад», клавіатура заміниться на основну. Обравши будь-яку підказку, відповідна інформація з’явиться у підпису до зображення (рис. 4.14).

Рисунок 4.14 – Використано підказку «Нація»

Спробувавши повторно використати цю ж підказку, користувач отримає відповідне сповіщення (рис. 4.15).

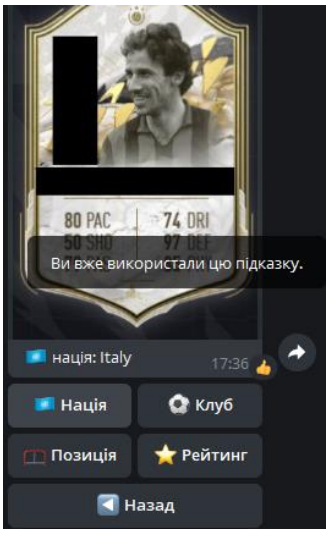

Рисунок 4.15 – Користувач намагається повторно використати підказку

Інформація з інших підказок буде виводитись у підписі з нового рядка, як показано на рисунку 4.16.

Рисунок 4.16 – Використано другу підказку

Якщо користувач натиснув кнопку «Вгадати» система замінює клавіатуру на іншу, де є варіанти відповіді і також кнопка «Назад» (рис. 4.17).

Рисунок 4.17 – Користувач натиснув кнопку «Вгадати»

У випадку, коли користувач відповів правильно, підпис під зображенням змінюється, клавіатура видаляється, після чого чат-бот надсилає наступну картку з основною клавіатурою (рис. 4.18).

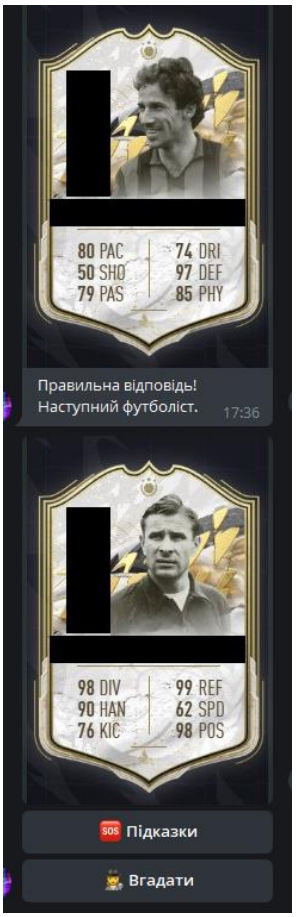

Рисунок 4.18 – Користувач відповів правильно

У випадку, коли відповідь неправильна, підпис змінюється на відповідний, з інформацією про кількість набраних очок, як на рисунку 4.19. На цьому ігровий процес завершується, кінцевий автомат переходить у фінішний стан та припиняє свою роботу.

Рисунок 4.19 – Користувач не вгадав

4.2 Засоби безпеки при функціонуванні системи

У чат-боті передбачено дві команди для адміністратора:

– download;

– delete.

Їх необхідно зробити недоступними для інших користувачів. Так, перед тим, як виконувати основний функціонал команд, система зчитує id користувача, який намагається запустити команду, і у випадку, коли id не співпадає з id адміністратора, бот відправляє повідомлення, що ця команда доступна лише адміністратору.

Id адміністратора, як і токен для взаємодії з Telegram Bot API винесено у окремий конфігураційний файл option.py.

Відправляючи команду download через пробіл слід вказати число – кількість сторінок, які алгоритм веб-скрапінгу повинен опрацювати. Якщо число не вказано або введено некоректно, спрацьовує механізм обробки виключень. Це дозволяє системі продовжувати працювати, і не виникає необхідності перезапускати бот.

Механізми обробки виключень також передбачені у функції, яка додає нового користувача у базу даних, на випадок, якщо користувач вдруге запустив команда start.

У алгоритмі веб-скрапінгу реалізована перевірка кількості зчитаних рядків таблиці, оскільки поточна сторінка може виявитися порожньою. Механізм обробки виключень також необхідний, оскільки в одного футболіста може бути декілька різних типів карток, а до бази даних додається лише одна картка для кожного футболіста.

4.3 Висновки за розділом

У поточному розділі було детально описано роботу створеного чат-боту. Описано роботу усіх команд та кнопок. Було детально описано роботу системи

при різних послідовностях дій користувача. Також було продемонстровано зовнішній вигляд інтерфейсу та скріншоти взаємодії з чат-ботом.

Також було описано механізм захисту бази даних та зображень карток від усіх користувачів, який передбачає перевірку id користувача на відповідність id адміністратора.

У деяких функціях модулів передбачено обробку виключень, які можуть виникати. Цей механізм дозволяє чат-боту працювати без необхідності перезапуску у разі виникнення критичного виключення.

Для безпеки токен взаємодії з Telegram Bot API та id адміністратора винесено в окремий конфігураційний файл.

5 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

5.1 Модульне тестування

При розробці програмних додатків необхідно перевіряти коректність та стабільність роботи програми. Це полегшить підтримку програми після здачі проекту замовнику у роботу. Оскільки виявити помилки та виправити їх легше на етапі, коли програма знаходиться у розробці, ніж в ситуації, коли реліз відбувся, і необхідне термінове втручання розробника.

Для перевірки роботи програми існують різні методи тестування, одним з таких методів є модульне тестування [29].

Модульне тестування передбачає проведення тестування кожного модуля програмної системи окремо.

У мові python існує бібліотека unittest, яка призначена для написання тестів розроблюваних модулів.

5.1.1 Тестування алгоритму веб-скрапінгу

Адміністратор чат-боту має команду завантаження інформації про футболістів у базу даних та зображень на локальну машину. Обробник команди викликає функцію веб-скрапінгу і передає на вході параметр – кількість сторінок для обробки.

Оскільки адміністратор вводить вручну кількість сторінок, то може бути введене від’ємне число, нуль чи взагалі рядок символів. Для коректної роботи функціонала чат-боту, необхідно провести тестування функції веб-скрапінгу й виявити можливі помилки.

Необхідно протестувати функцію на виникнення виключень помилка значень та типу – ValueError і TypeError відповідно.

Так, було розроблено 2 тести, які перевіряють роботу функції парсингу, лістинг міститься у додатку В.9.

Результат тестування на рисунку 5.1. Функція ніяк не оброблює можливі помилки.

```
======================================================================
FAIL: test_values (test_parse_cards.TestParse)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\ekote\Desktop\gpbc_bot\test_parse_cards.py", line 7, in test_values
    self.assertRaises(ValueError, parse, -5)
AssertionError: ValueError not raised by parse

----------------------------------------------------------------------
Ran 2 tests in 0.022s

FAILED (failures=1, errors=1)
```

Рисунок 5.1 – Результати тестування функції parse()

Функція parse викликається обробником команди download, в програмному коді обробника передбачена обробка цих помилок заздалегідь, таким чином, у разі виникнення виключень, функція parse не викликається взагалі. Можна зробити висновок, що система відпрацьовує у цьому випадку коректно.

Механізм обробки можливих виключень приведено на рисунку 5.2.

```
try:
    num_page = message.text.split()[1]
    if type(num_page) not in [int]:
        raise TypeError
    if num_page <= 0:
        raise ValueError
except TypeError as te:
    print(te)
    await message.answer("Спробуй ще раз. Поруч з командою необхідно ввести позитивне цілочисельне значення.\n\n"
                         "Приклад:\n\n"
                         "/download 5")
except ValueError as ve:
    print(ve)
    await message.answer("Спробуй ще раз. Поруч з командою необхідно ввести позитивне цілочисельне значення.\n\n"
                         "Приклад:\n\n"
                         "/download 5")
except IndexError as ie:
    print(ie)
    await message.answer("Спробуй ще раз. Поруч з командою напиши кількість сторінок для обробки.\n\n"
                         "Приклад:\n\n"
                         "/download 5")
else:
    await message.answer("Завантаження карток до бази даних почалося.")
    parse_cards.parse(num_page)
    await message.answer("Завантаження завершено.")
```

Рисунок 5.2 – Механізм обробки виключень

5.1.2 Тестування алгоритму рандомайзера

Ігровий процес передбачає випадковий вибір чотирьох футболістів з бази даних. У якості аргументу, функція вибору інформації з бази даних, приймає випадково-згенероване число. Для ігрового процесу необхідно згенерувати 4 таких числа, оскільки 4 варіанти відповідей запропоновано користувачу.

Для генерації чисел використовується функція, яка випадковим чином генерує 4 унікальних, цілих числа з запропонованого діапазону, тобто від 1 і до кількості записів у таблиці footballer.

У процесі функціонування чат-боту, може виникнути ситуація, коли адміністратор очистив базу даних, але нова інформація ще не додана до база даних. У цей час якийсь користувач захоче пограти, виклик команди game спричинить аварійну зупинку роботи чат-боту, і потрібно буде перезапустити бот.

Для функції рандомайзера було написано unit-тести, які перевіряє правильність типу даних, які повертає функція, та роботу програму, у випадку, коли правою межею генерування буде 0, тобто в таблиці footballer немає жодного запису.

Обидва тести пройшли успішно (рис. 5.3). Це обумовлено тим, що функція random.randint, яка використовується у генеруванні одного числа викликає необхідне виключення, якщо права межа менша, ніж ліва.

Рисунок 5.3 – Результати тестування функції randUnique4num

Але робота чат-боту у такому випадку аварійно припиняється, і необхідно його запустити знову. Для продовження роботи необхідно додати механізм обробки виключення ValueError (рис. 5.4).

```
try:
    await FSMGame.start_game.set()
    num = randUnique4num(1, sqlite_db.sql_count_footballer())
except ValueError as VE:
    print(VE)
    print('База даних порожня')
    await message.answer("⌛ Вибачте, гра тимчасово не працює.\n✍   Зверніться до автора: @kotinho7.")
    await state.finish()
else:
    cards = sqlite_db.sql_random_footballer(num[0])
    async with state.proxy() as data:
        data['score'] = 0
        data['1'] = cards[0]
        data['pos'] = cards[1]
        data['country'] = cards[2]
        data['club'] = cards[3]
        data['rate'] = cards[4]
        data['path'] = cards[5]
        data['p1'] = 0
        data['p2'] = 0
        data['p3'] = 0
        data['p4'] = 0
        data['2'] = sqlite_db.sql_random_footballer(num[1])[0]
        data['3'] = sqlite_db.sql_random_footballer(num[2])[0]
        data['4'] = sqlite_db.sql_random_footballer(num[3])[0]
    await message.answer_photo(open(data['path'], 'rb'), reply_markup=game_KB.ilkb)
    await FSMGame.game.set()
```

Рисунок 5.4 – Механізм обробки виключення ValueError

5.2 Висновки за розділом

У цьому розділі було описано метод модульного тестування. Цей метод дозволяє швидко виявити і виправити помилки на етапі розробки програмного забезпечення, що в свою чергу скорочує час розробки програмних продуктів та полегшує їх обслуговування в подальшому.

Написано чотири тести для функції веб-скрапінгу та функції randUnique4num.

У результаті тестування було виявлено помилки у роботі функції парсингу, але механізм їх обробки було завчасно розроблено в функції download, яка й викликає парсер.

Тести функції райндомайзера завершено успішно, оскільки виключення викликається функцією random.randint.

Після цього було вручну перевірено випадок, коли база даних футболістів пуста, й відбулася спроба запуску ігрового процесу.

Так, незважаючи на те, що виключення генерується, робота чат-боту припиняється, і його необхідно перезапускати. Така ситуація виникає, коли запущена команда game, але в таблиці footballer немає жодного запису, тобто ліва межа генерування чисел 0.

Для усунення аварійного припинення роботи боту додатково було написано механізм обробки виключення ValueError.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи проведено аналіз предметної області, наведено обґрунтування обраної теми, визначену мету, об’єкт, предмет та методи дослідження. Також проведено аналіз конкурентів.

Проаналізувавши предметну область наведено теоретичні відомості, які необхідні для розробки чат-боту. Так, було розглянуто такі теоретичні положення:

– використання мови UML у проєктуванні програмних продуктів;

– принципи та положення модульного та об’єкто-орієнтовного програмування;

– технологія отримання даних веб-скрапінг;

– розмітка веб-сторінок;

– синтаксичне дерево розбору;

– патерни проєктування;

– теорія кінцевих автоматів.

Було сформовано завдання на кваліфікаційну роботу:

– упорядкований збір інформації про футболістів;

– зберігання інформації у спроєктованій базі даних;

– використання отриманої інформації про футболістів в ігровому процесі;

– обробка зображень при зборі інформації;

– створення машини станів для реалізації ігрового процесу.

Після завершення аналітичної роботи було виконано проєктування системи, в ході якого було обрано практичні засоби для виконання поставлених завдань та розроблено архітектуру системи. Таким чином, було розроблено UML-діаграми:

– діаграма варіантів використання для актора «Адміністратор»;

– діаграма варіантів використання для актора «Гравець»;
– діаграма діяльності запуску команди /download;
– діаграма діяльності запуску команди /delete;
– діаграма кооперації запуску команди /download;
– діаграма станів системи під час ігрового процесу;
– діаграма послідовностей для випадку невірної відповіді;
– діаграма класів машини станів;
– діаграма пакетів;
– діаграма розміщення системи.

В ході розробки чат-боту було описано особливості реалізації модулів системи, наведено структуру бази даних, побудовано блок-схему алгоритму веб-скрапінгу.

Після завершення розробки було розроблено unit-тести для алгоритму веб-скрапінгу та рандомайзера.

Тести веб-скрапінгу виявили помилки, але механізм їх обробки був написаний на етапі розробки.

Тести рандомайзеру помилок не виявили, але в ході ручного тестування виявлено аварійне завершення роботи чат-боту, при спробі запустити ігровий процес, якщо таблиця footballer пуста. Виявлений недолік було усунуто механізмом обробки відповідного виключення.

В ході виконання кваліфікаційної роботи було виконано усі поставлені задачі:

– реалізовано алгоритм веб-скрапінгу;
– розроблена база даних для зберігання інформації;
– розроблено механізм обробки зображень;
– застосовано патерн проєктування «Стан» для реалізації ігрового процесу шляхом створення кінцевого автомату.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Python [Електронний ресурс] – Режим доступу: https://www.python.org.

2. Aiogram [Електронний ресурс] – Режим доступу: https://docs.aiogram.dev/en/latest/.

3. SQLite3 [Електронний ресурс] – Режим доступу: https://docs.python.org/3/library/sqlite3.html.

4. Мейер Б. Основы объектно-ориентированного программирования / Б. Мейер. – Национальный Открытый Университет «ИНТУИТ», 2016 – 970 с.

5. Программирование на Python 3. Подробное руководство. – Пер. с англ. – СПб.: Символ-Плюс, 2009. – 608 с.

6. Брауэр В. Введение в теорию конечных автоматов / В. Брауэр. – М.: Радио и связь, 1987. — 392 с.

7. Стивенс Р. Алгоритмы. Теория и практическое применение / Р. Стивенс. – Москва : Издательство «Э», 2016. – 54с.

8. Программирование компьютерного зрения на языке Python. / пер. С англ. Слинкин А. А. – М.: ДМК Пресс, 2016. -312 с.:

9. GF Bot(Guess The Footballer) [Електронний ресурс] – Режим доступу: https://t.me/footGuessrbot.

10. Квиззери | Угадай футболиста [Електронний ресурс] – Режим доступу: https://t.me/GuessFootball.

11. Митчелл Р. Скрапинг веб-сайтов с помощью Python / Р. Митчелл., пер. с англ. А. В. Груздев. – М.: ДМК Пресс, 2016. – 280 с.

12. Фрімен Е. Head First. Патерни проєктування / Е. Фрімен, Е. Робсон, Б. – Фабула, 2020. — 672 с.

13. Стан [Електроний ресурс] – Режим доступу: https://refactoring.guru/uk/design-patterns/state.

14. Буч Г. Язык UML. Руководство пользователя / Г. Буч, Дж. Рамбо, А. Джекобсон. – М. : ДМК, 2013. – 432 с.

15. Ташков П. А. Веб-мастеринг на 100 %: HTML, CSS, JavaScript, PHP, CMS, AJAX, раскрутка. / П. А. Ташков. — СПб.: Питер, 2010. — 512 с.

16. Дмитрієва О. А. Методичні вказівки і завдання до виконання практичних, розрахункових і самостійних робіт за курсом «Теорія синтаксичного аналізу і компіляції» для студентів спеціальності 121 Інженерія програмного забезпечення всіх форм навчання / О. А. Дмитрієва. – Покровськ: ДонНТУ, 2020 – 78 с.

17. Лафоре Р. Объектно-ориентированное программирование в C++ / Р. Лафоре. – Питер, 2015. – 928 с.

18. Python-telegram-bot [Електронний ресурс] – Режим доступу: https://python-telegram-bot.readthedocs.io/en/v20.0a0/.

19. Боггс У. UML и Rational Rose / У. Боггс, М. Боггс. – М.: "ЛОРИ", 2000. – 582 с.

20. Анализ и проектирование информационных систем с помощью UML 2.0, 3-е изд. : Пер. с англ. — М. : ООО “И.Д. Вильямс”, 2008. — 816 с.

21. UML. Основы, 2е издание. – Пер. с англ. – СПб: Символ-Плюс, 2002. – 192 с.

22. UML. Основы, 3е издание. – Пер. с англ. – СПб: Символ-Плюс, 2004. – 192 с.

23. DB Browser [Електронний ресурс] – Режим доступу: https://sqlitebrowser.org.

24. BeautifulSoup [Електронний ресурс] – Режим доступу: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

25. HTML2Image [Електронний ресурс] – Режим доступу: https://github.com/vgalin/html2image.

26. Python image library [Електронний ресурс] – Режим доступу: https://omz-software.com/pythonista/docs/ios/PIL.html.

27. PyCharm [Електронний ресурс] – Режим доступу: https://www.jetbrains.com/ru-ru/pycharm/.

28. DrawIO [Електронний ресурс] – Режим доступу: https://www.diagrams.net.

29. Саммерфилд М. Программирование на Python 3. Подробное руководство / М. Саммерфилд. – Пер. с англ. – СПб.: Символ-Плюс, 2009. – 608 с., ил.

Додаток А

Зауваження нормоконтролера

Таблиця А.1 – Зауваження нормоконтролера

Сторінка	Позначення	Зміст зауваження
ПЗ		Нещільне заповнення сторінок ПЗ
ПЗ		Перед рисунками нема порожнього рядка
С. 70-72		Помилки в оформленні використаних джерел

Додаток Б

Графічна частина

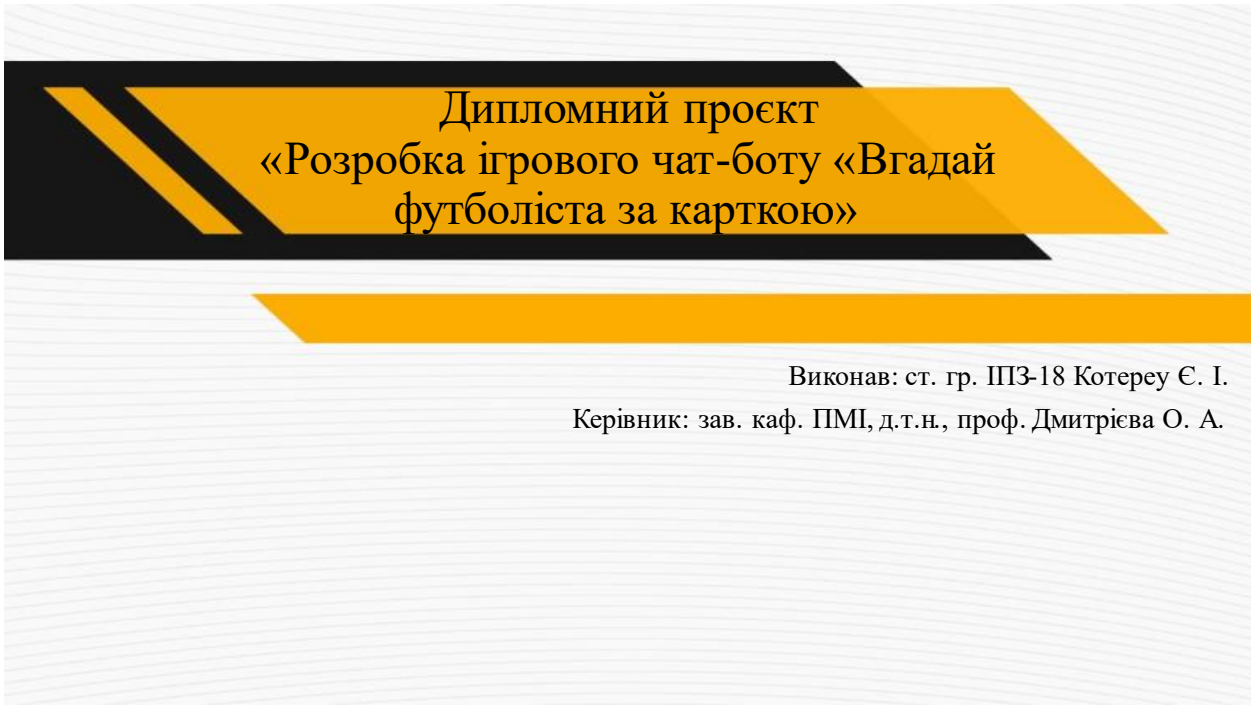

Рисунок Б.1 – Титульний слайд

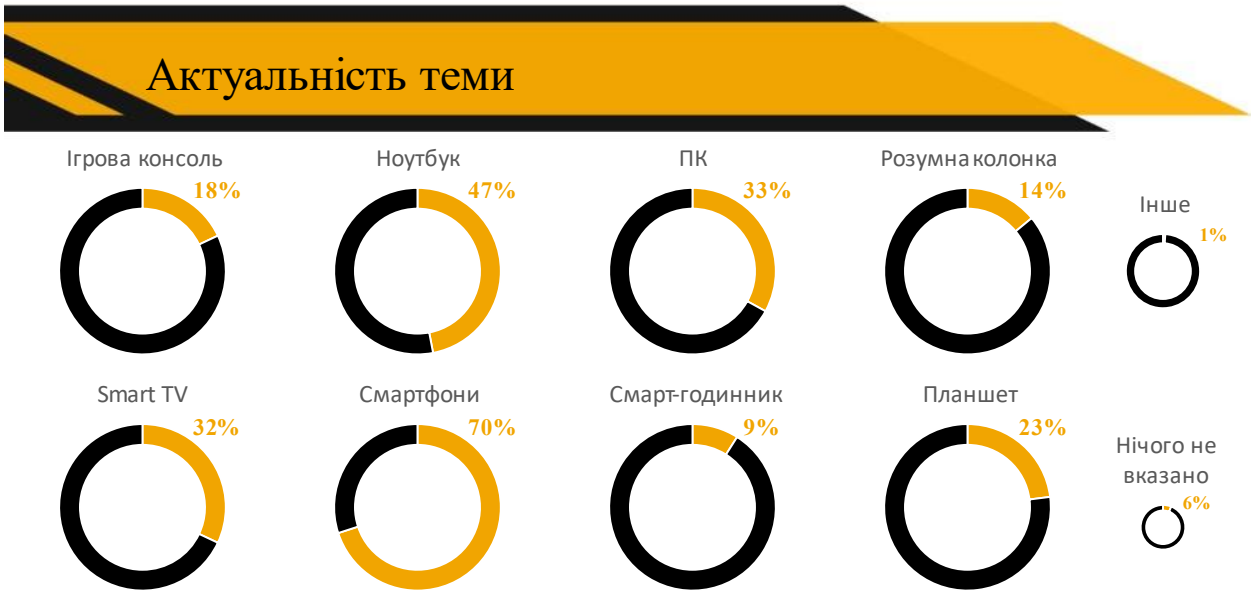

Рисунок Б.2 – Актуальність теми

Мета та завдання дослідження

Мета роботи: розробка сценарію та логіки ігрового чат-боту «Вгадай футболіста за карткою» для месенджера Telegram засобами мови Python, бібліотеки Aiogram , бази даних SQLite .

Завдання дослідження: створення простої і цікавої гри з вільним доступом, яка не потребує інсталяції на пристрій, тобто не займає пам’ять девайсу.

Рисунок Б.3 – Мета та завдання дослідження

Об’єкт і предмет дослідження

Об’єкт дослідження: процеси проєктування, розробки та тестування ігрового чат-боту у месенджері Telegram .

Предмет дослідження: алгоритми та методи розробки ігрового чат-боту, механізми взаємодії з Telegram Bot API, які реалізовані у бібліотеці Aiogram .

Рисунок Б.4 – Об’єкт і предмет дослідження

Основні завдання

- упорядкований збір інформації про футболістів;
- зберігання інформації у спроєктованій базі даних;
- використання отриманої інформації про футболістів в ігровому процесі;
- обробка зображень при зборі інформації;
- створення машини станів для реалізації ігрового процесу.

Рисунок Б.5 – Основні завдання

Інструментарій та технології для розробки

Для розробки програмного продукту використано теоретичні знання з:

- теорії алгоритмів;
- теорії кінцевих автоматів;
- теорії обробки зображень;
- об’єктно-орієнтовного та модульного програмування.

Практичні методи вирішення поставлених завдань:

- технологія longpolling ;
- функціонал бібліотеки Aiogram для взаємодії з Telegram Bot API;
- модуль SQLite3 для роботи з базою даних;
- можливості модуля BeatifulSoup4 для парсингу сайтів;
- бібліотеки html2image, PIL для обробки зображень.

Рисунок Б.6 – Інструментарій та технології для розробки

Патерн проєктування. Діаграма класів

Для реалізації логіки ігрового процесу було застосовано патерн проєктування «Стан» та відповідно, розроблено кінцевий автомат з пам'яттю.

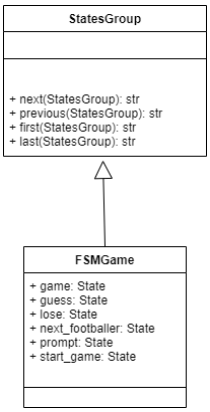

Рисунок Б.7 – Патерн проєктування. Діаграма класів

Діаграми варіантів використання

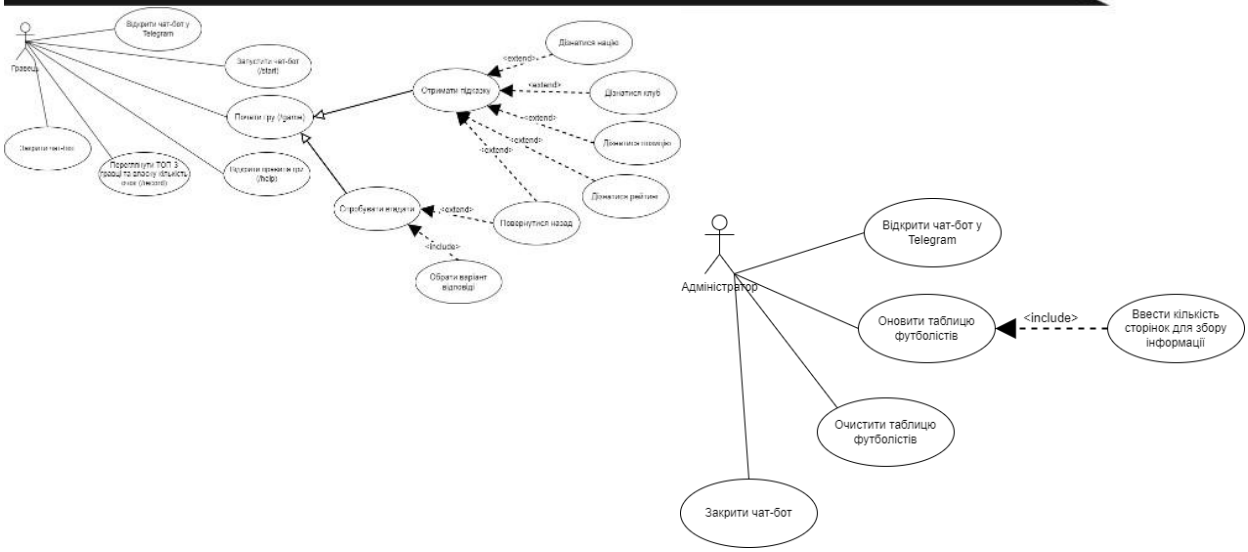

Рисунок Б.8 – Діаграми варіантів використання

Блок-схема алгоритму веб-скрапінгу

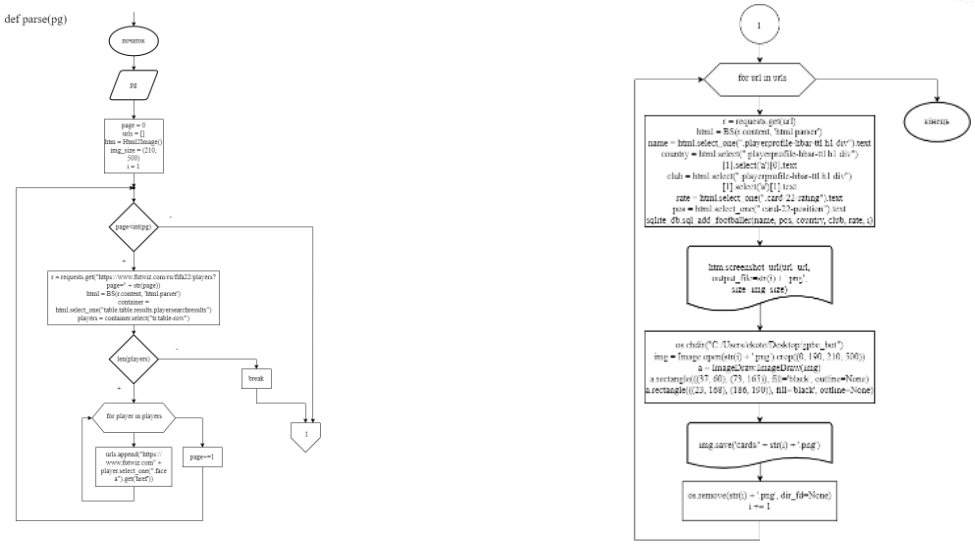

Рисунок Б.9 – Блок-схема алгоритму веб-скрапінгу

Розроблені модулі. Діаграма пакетів

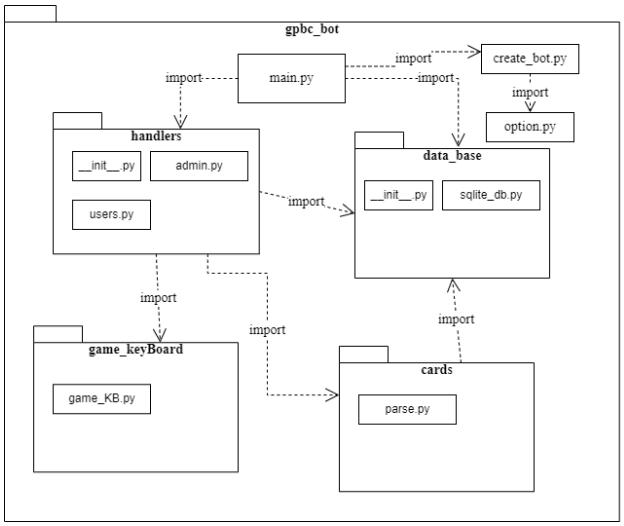

Рисунок Б.10 – Розроблені модулі. Діаграма пакетів

Структура бази даних

footballer		CREATE TABLE footballer(id integer PRIMARY KEY,name text,position text,country text,club text,rate integer,path text,UNIQUE(name))
id	integer	"id" integer
name	text	"name" text
position	text	"position" text
country	text	"country" text
club	text	"club" text
rate	integer	"rate" integer
path	text	"path" text
user_score		CREATE TABLE user_score(user_id integer PRIMARY KEY,first_name text,last_name text,user_name text,score integer,UNIQUE(user_id))
user_id	integer	"user_id" integer
first_name	text	"first_name" text
last_name	text	"last_name" text
user_name	text	"user_name" text
score	integer	"score" integer

Рисунок Б.11 – Структура бази даних

Результати роботи

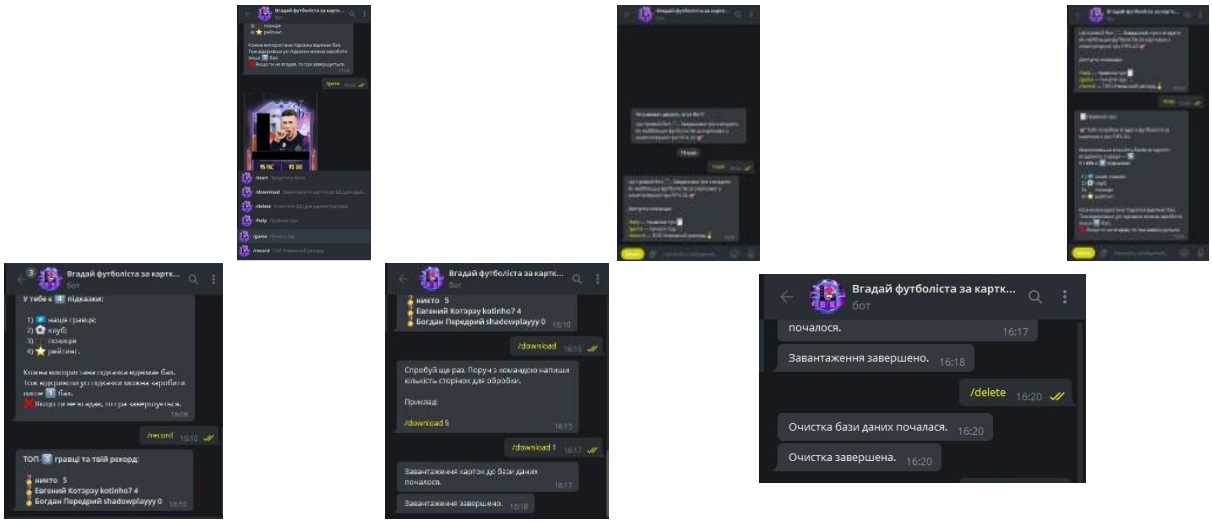

Рисунок Б.12 – Результати роботи

Результати роботи

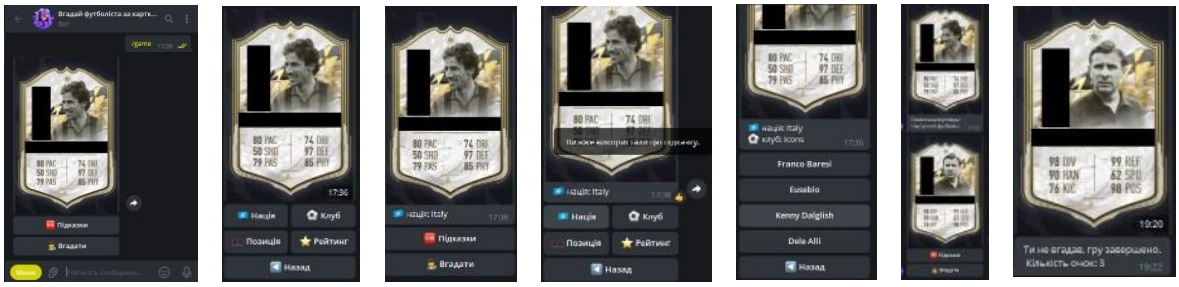

Рисунок Б.13 – Результати роботи, продовження

Напрями вдосконалення програмної системи

Для вдосконалення програмної системи, її можна розмістити на платному сервері. При розміщенні на сервері доцільно використати замість технології longpolling – технологію webhook. Ці заходити дозволять працювати боту цілодобово, покращиться стійкість роботи та швидкодія.

Також можна розширити функціонал чат-боту, розробивши нові режими гри – «Вгадай клуб за прізвиськом» та «Вгадай клуб за формою».

Рисунок Б.14 – Напрями вдосконалення програмної системи

Висновки

У ході виконання кваліфікаційної роботи проведено аналіз предметної області, її актуальності, визначено мету, об’єкт, предмет та методи дослідження.

Було наведено необхідні теоретичні відомості та технології, необхідні для розробки.

При проєктуванні було розроблено UML-діаграми, використано патерн проєктування «Стан».

У результаті розробки було створено усі необхідні модулі, усі поставлені завдання успішно виконані.

Також було розглянуто можливості для подальшого вдосконалення програмної системи. Наведено технології, які можна використати для покращення стійкості та швидкості реагування. Окрім цього розглянуто й розширення функціональних можливостей чат-боту.

Рисунок Б.15 - Висновки

Додаток В.1

Лістинг модуля створення екземпляру бота

```
#create_bot.py
from aiogram import Bot
from aiogram.dispatcher import Dispatcher
from aiogram import types
import option
from aiogram.contrib.fsm_storage.memory import MemoryStorage


storage = MemoryStorage()
bot = Bot(option.TOKEN)
dp = Dispatcher(bot, storage=storage)


async def set_default_commands(dp):
    await dp.bot.set_my_commands([
        types.BotCommand("start", "Запустити бота"),
        types.BotCommand("download", "Завантажити картки до БД (для адміністратора)"),
        types.BotCommand("delete", "Очистити БД (для адміністратора)"),
        types.BotCommand("help", "Правила гри"),
        types.BotCommand("game", "Почати гру"),
        types.BotCommand("record", "ТОП-3+власний рекорд"),
    ])
```

Додаток В.2

Лістинг модуля запуску роботи бота

```
#main.py
from aiogram.utils import executor
from create_bot import set_default_commands
from create_bot import dp
from data_base import sqlite_db
from handlers import admin, users


async def on_startup(_):
    print('Bot is working')
    sqlite_db.sql_start()
    await set_default_commands(dp)


users.register_handlers_admin(dp)
admin.register_handlers_admin(dp)


if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True, on_startup=on_startup)
```

Додаток В.3

Лістинг модуля веб-скрапінгу

```
#cards/parse_cards.py
import requests
from bs4 import BeautifulSoup as BS
from html2image import Html2Image
from PIL import Image, ImageDraw
from data_base import sqlite_db
import os
import sqlite3 as sq


def parse(pg):
    page = 0
    urls = []
    htm = Html2Image()
    img_size = (210, 500)
    i = 1
    while page < int(pg):
        r = requests.get("https://www.futwiz.com/ru/fifa22/players?page=" +
str(page))
        html = BS(r.content, 'html.parser')
        container = html.select_one("table.table.results.playersearchresults")
        players = container.select("tr.table-row")
        if len(players):
            for player in players:
                urls.append("https://www.futwiz.com" + player.select_one(".face
a").get('href'))
            page += 1
        else:
            break
```

```
    for url in urls:
        try:
            r = requests.get(url)
            html = BS(r.content, 'html.parser')
            name = html.select_one(".playerprofile-hbar-ttl h1 div").text
            country = html.select(".playerprofile-hbar-ttl h1 div")[1].select('a')[0].text
            club = html.select(".playerprofile-hbar-ttl h1 div")[1].select('a')[1].text
            rate = html.select_one(".card-22-rating").text
            pos = html.select_one(".card-22-position").text
            sqlite_db.sql_add_footballer(name, pos, country, club, rate, i)
            htm.screenshot_url(url=url, output_file=str(i) + '.png', size=img_size)
            os.chdir("C:/Users/ekote/Desktop/gpbc_bot")
            img = Image.open(str(i) + '.png').crop((0, 190, 210, 500))
            a = ImageDraw.ImageDraw(img)
            a.rectangle(((37, 60), (73, 165)), fill='black', outline=None)
            a.rectangle(((23, 168), (186, 190)), fill='black', outline=None)
            img.save('cards/' + str(i) + '.png')
            os.remove(str(i) + '.png', dir_fd=None)
            i += 1
        except sq.IntegrityError as nf:
            print(nf)


def delete():
    count = sqlite_db.sql_count_footballer()
    sqlite_db.sql_del_footballer()
    os.chdir('cards')
    for i in range(count):
        os.remove(str(i + 1) + '.png', dir_fd=None)
```

Додаток В.4

Лістинг модуля взаємодії з базою даних

```
#data_base/sqlite_db.py
import sqlite3 as sq


def sql_start():
    global base, cur
    base = sq.connect('gpbc_bot.db')
    cur = base.cursor()
    if base:
        print('Database connected.')
    base.execute('CREATE TABLE IF NOT EXISTS user_score('
                 'user_id integer PRIMARY KEY,'
                 'first_name text,'
                 'last_name text,'
                 'user_name text,'
                 'score integer,'
                 'UNIQUE(user_id)'
                 ');')
    base.commit()
    base.execute('CREATE TABLE IF NOT EXISTS footballer('
                 'id integer PRIMARY KEY,'
                 'name text,'
                 'position text,'
                 'country text,'
                 'club text,'
                 'rate integer,'
                 'path text,'
                 'UNIQUE(name)'
                 ');')
    base.commit()
```

```
def sql_add_user(user_id, first_name, last_name, username):
    try:
        cur.execute('INSERT INTO user_score
(user_id,first_name,last_name,user_name,score) VALUES(?,?,?,?,?);',
                    (user_id, first_name, last_name, username, 0))
        base.commit()
    except sq.IntegrityError as nu:
        print(nu)


def sql_add_footballer(name, pos, country, club, rate, i):
    cur.execute('INSERT INTO footballer (id,name,position,country,club,rate,path)
VALUES(?,?,?,?,?,?,?);',
                (i, name, pos, country, club, rate, 'cards/' + str(i) + '.png'))
    base.commit()


def sql_del_footballer():
    cur.execute('DELETE FROM footballer')
    base.commit()


def sql_record_user(user_id):
    top = cur.execute(
        'SELECT first_name, last_name, user_name,score FROM user_score ORDER
BY score desc limit 3').fetchall()
    you = cur.execute('SELECT first_name, last_name, user_name,score FROM
user_score WHERE user_id==?',
```

```
                (user_id,)).fetchone()
    if you not in top:
        top.append(you)
    return top


def sql_count_footballer():
    return len(cur.execute('SELECT * FROM footballer').fetchall())


def sql_random_footballer(num_id):
    return cur.execute('SELECT name,position,country,club,rate, path from
footballer WHERE id==?', (num_id,)).fetchone()


def sql_update_record(score, user_id):
    cur.execute('UPDATE user_score SET score ==? WHERE user_id==?', (score,
user_id))
    base.commit()


def sql_get_record(user_id):
    return cur.execute('SELECT score FROM user_score WHERE user_id==?',
(user_id,)).fetchone()
```

Додаток В.5

Лістинг модуля для обробки команд адміністратора

```
#handlers/admin.py
from aiogram import types, Dispatcher
from cards import parse_cards
import option


Adm_id = option.ADMIN_ID


# @dp.message_handler(commands=['download'])
async def download(message: types.Message):
    if message.from_user.id == Adm_id:
        try:
            num_page = int(message.text.split()[1])
            if num_page <= 0:
                raise ValueError
        except ValueError as ve:
            print(ve)
            await message.answer("Спробуй ще раз. Поруч з командою необхідно
ввести позитивне цілочисельне значення.\n\n"
                                 "Приклад:\n\n"
                                 "/download 5")
        except IndexError as ie:
            print(ie)
            await message.answer("Спробуй ще раз. Поруч з командою напиши
кількість сторінок для обробки.\n\n"
                                 "Приклад:\n\n"
                                 "/download 5")
        else:
            await message.answer("Завантаження карток до бази даних почалося.")
```

```
            parse_cards.parse(num_page)
            await message.answer("Завантаження завершено.")
    else:
        await message.answer("Ця команда доступна лише адміністратору.")


# @dp.message_handler(commands=['delete'])
async def delete(message: types.Message):
    if message.from_user.id == Adm_id:
        await message.answer("Очистка бази даних почалася.")
        parse_cards.delete()
        await message.answer("Очистка завершена.")
    else:
        await message.answer("Ця команда доступна лише адміністратору.")


# @dp.message_handler()
async def echo_send(message: types.Message):
    await message.answer("Бот не приймає повідомлення. Для роботи з ботом використовуй команди та кнопки.")
    await message.delete()


def register_handlers_admin(dp: Dispatcher):
    dp.register_message_handler(download, commands=['download'])
    dp.register_message_handler(delete, commands=['delete'])
    dp.register_message_handler(echo_send)
```

Додаток В.6

Лістинг модуля для обробки загальних команд користувачів

```
#handlers/users.py
from aiogram import types, Dispatcher
from data_base import sqlite_db
from aiogram.dispatcher import FSMContext
from aiogram.dispatcher.filters.state import State, StatesGroup
from aiogram.dispatcher.filters import Text
import random
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton
from game_keyBoard import game_KB


class FSMGame(StatesGroup):
    start_game = State()
    prompt = State()
    game = State()
    guess = State()
    next_footballer = State()
    lose = State()


def randUnique4num(a, b):
    result = []
    seen = set()
    for i in range(4):
        x = random.randint(a, b)
        while x in seen:
            x = random.randint(a, b)
        seen.add(x)
        result.append(x)
    return result
```

```
# @dp.message_handler(commands=['start'])
async def start(message: types.Message):
    await message.answer(
        "Це ігровий бот🎮. Завданням гри є вгадати як найбільше футболістів за картками з комп'ютерної гри FIFA 22.🎯\n\n"
        "Доступні команди:\n\n"
        "/help — правила гри📜\n"
        "/game — почати гру🎮\n"
        "/record — ТОП-3+власний рекорд🏅")
    first_name = message.from_user.first_name
    last_name = message.from_user.last_name
    username = message.from_user.username
    user_id = message.from_user.id
    if last_name is None:
        last_name = ''
    if username is None:
        username = ''
    sqlite_db.sql_add_user(user_id, first_name, last_name, username)


# @dp.message_handler(commands=['help'])
async def help(message: types.Message):
    await message.answer("📜Правила гри:\n\n"
                         "🎯Тобі потрібно вгадати футболіста за карткою з гри FIFA 22.\n\n"
```

"Максимальна кількість балів за одного вгаданого гравця — 5.\n"

"У тебе є 4 підказки:\n\n"

" 1) UN нація гравця;\n"

" 2) ⚽ клуб;\n"

" 3) 🥅 позиція\n"

" 4) ⭐ рейтинг.\n\n"

"Кожна використана підказка віднімає бал. Тож відкривши усі підказки можна заробити лише 1 бал.\n"

"❌ Якщо ти не вгадав, то гра завершується.",
parse_mode='html')

```
# @dp.message_handler(commands=['game'],state=None)
async def game(message: types.Message, state: FSMContext):
    try:
        await FSMGame.start_game.set()
        num = randUnique4num(1, sqlite_db.sql_count_footballer())
    except ValueError as VE:
        print(VE)
        print('База даних порожня')
        await message.answer("⏳ Вибачте, гра тимчасово не працює.\n✍️ Зверніться до автора: @kotinho7.")
        await state.finish()
    else:
        cards = sqlite_db.sql_random_footballer(num[0])
        async with state.proxy() as data:
            data['score'] = 0
```

```
            data['1'] = cards[0]
            data['pos'] = cards[1]
            data['country'] = cards[2]
            data['club'] = cards[3]
            data['rate'] = cards[4]
            data['path'] = cards[5]
            data['p1'] = 0
            data['p2'] = 0
            data['p3'] = 0
            data['p4'] = 0
            data['2'] = sqlite_db.sql_random_footballer(num[1])[0]
            data['3'] = sqlite_db.sql_random_footballer(num[2])[0]
            data['4'] = sqlite_db.sql_random_footballer(num[3])[0]
        await message.answer_photo(open(data['path'], 'rb'),
reply_markup=game_KB.ilkb)
        await FSMGame.game.set()


# @dp.callback_query_handlers(text='prompt',state=FSMGame.game)
async def prompt_cbq(callback: types.CallbackQuery):
    await callback.message.edit_reply_markup(reply_markup=game_KB.pkb)
    await FSMGame.prompt.set()
    await callback.answer()


# @dp.callback_query_handlers(text='nation',state=FSMGame.prompt)
async def nat_cbq(callback: types.CallbackQuery, state: FSMContext):
    cap = ''
    if callback.message.caption is not None:
        cap = callback.message.caption + '\n'
```

```
    async with state.proxy() as data:
        if data['p1'] == 0:
            await callback.message.edit_caption(cap + 'ᴜɴ нація: ' + data['country'],
reply_markup=game_KB.ilkb)
            data['p1'] = 1
        else:
            await callback.answer('Ви вже використали цю підказку.')
            await callback.message.edit_reply_markup(reply_markup=game_KB.pkb)
    await FSMGame.game.set()


# @dp.callback_query_handlers(text='position',state=FSMGame.prompt)
async def pos_cbq(callback: types.CallbackQuery, state: FSMContext):
    cap = ''
    if callback.message.caption is not None:
        cap = callback.message.caption + '\n'
    async with state.proxy() as data:
        if data['p2'] == 0:
            await callback.message.edit_caption(cap + '🥅 позиція: ' + data['pos'],
reply_markup=game_KB.ilkb)
            data['p2'] = 1
        else:
            await callback.answer('Ви вже використали цю підказку.')
            await callback.message.edit_reply_markup(reply_markup=game_KB.pkb)
    await FSMGame.game.set()


# @dp.callback_query_handlers(text='club',state=FSMGame.prompt)
async def club_cbq(callback: types.CallbackQuery, state: FSMContext):
```

```
    cap = ''
    if callback.message.caption is not None:
        cap = callback.message.caption + '\n'
    async with state.proxy() as data:
        if data['p3'] == 0:
            await callback.message.edit_caption(cap + '⚽ клуб: ' + data['club'],
reply_markup=game_KB.ilkb)
            data['p3'] = 1
        else:
            await callback.answer('Ви вже використали цю підказку.')
            await callback.message.edit_reply_markup(reply_markup=game_KB.pkb)
    await FSMGame.game.set()


# @dp.callback_query_handlers(text='rate',state=FSMGame.prompt)
async def rate_cbq(callback: types.CallbackQuery, state: FSMContext):
    cap = ''
    if callback.message.caption is not None:
        cap = callback.message.caption + '\n'
    async with state.proxy() as data:
        if data['p4'] == 0:
            await callback.message.edit_caption(cap + '⭐ рейтинг: ' +
str(data['rate']), reply_markup=game_KB.ilkb)
            data['p4'] = 1
        else:
            await callback.answer('Ви вже використали цю підказку.')
            await callback.message.edit_reply_markup(reply_markup=game_KB.pkb)
    await FSMGame.game.set()
```

```
# @dp.callback_query_handlers(text='guess',state=FSMGame.game)
async def guess_cbq(callback: types.CallbackQuery, state: FSMContext):
    y = randUnique4num(1, 4)
    gkb = InlineKeyboardMarkup(row_width=1)
    index = []
    vkb = ['', '', '', '']
    var = ['', '', '', '']
    async with state.proxy() as data:
        index.append(y.index(1))
        index.append(y.index(2))
        index.append(y.index(3))
        index.append(y.index(4))
        var[index[0]] = data['1']
        var[index[1]] = data['2']
        var[index[2]] = data['3']
        var[index[3]] = data['4']
    for i in range(4):
        vkb[index[i]] = InlineKeyboardButton(var[index[i]], callback_data=i + 1)
    gkb.add(vkb[0], vkb[1], vkb[2], vkb[3], game_KB.back)
    await callback.message.edit_reply_markup(reply_markup=gkb)
    await FSMGame.guess.set()


# @dp.callback_query_handlers(text='1',state=FSMGame.guess)
async def true_cbq(callback: types.CallbackQuery, state: FSMContext):
    await callback.message.edit_caption(caption="Правильна відповідь!
Наступний футболіст.", reply_markup=None)
    count = 0
    async with state.proxy() as data:
```

```
        if data['p1'] == 1:
            count += 1
        if data['p2'] == 1:
            count += 1
        if data['p3'] == 1:
            count += 1
        if data['p4'] == 1:
            count += 1
        data['score'] = data['score'] + (5 - count)
    await FSMGame.next_footballer.set()
    num = randUnique4num(1, sqlite_db.sql_count_footballer())
    cards = sqlite_db.sql_random_footballer(num[0])
    async with state.proxy() as data:
        data['1'] = cards[0]
        data['pos'] = cards[1]
        data['country'] = cards[2]
        data['club'] = cards[3]
        data['rate'] = cards[4]
        data['path'] = cards[5]
        data['p1'] = 0
        data['p2'] = 0
        data['p3'] = 0
        data['p4'] = 0
        data['2'] = sqlite_db.sql_random_footballer(num[1])[0]
        data['3'] = sqlite_db.sql_random_footballer(num[2])[0]
        data['4'] = sqlite_db.sql_random_footballer(num[3])[0]
    await callback.message.answer_photo(open(data['path'], 'rb'),
reply_markup=game_KB.ilkb)
    await FSMGame.game.set()
```

```
# @dp.callback_query_handlers(text='2,3,4',state=FSMGame.guess)
async def false_cbq(callback: types.CallbackQuery, state: FSMContext):
    async with state.proxy() as data:
        score = data['score']
    await callback.message.edit_reply_markup(reply_markup=None)
    await callback.message.answer(text='Ти не вгадав, гру завершено.\n Кількість
очок: ' + str(score))
    await FSMGame.lose.set()
    if score > sqlite_db.sql_get_record(callback.from_user.id)[0]:
        sqlite_db.sql_update_record(score, callback.from_user.id)
    await state.finish()


# @dp.callback_query_handlers(text='back',state=FSMGame.prompt)
async def back_cbq(callback: types.CallbackQuery):
    await callback.message.edit_reply_markup(reply_markup=game_KB.ilkb)
    await FSMGame.game.set()
    await callback.answer()


# @dp.message_handler(commands=['record'])
async def record(message: types.Message):
    user_id = message.from_user.id
    ans = 'ТОП-3 гравці та твій рекорд:\n\n'
    for people in sqlite_db.sql_record_user(user_id):
        ans = ans + '🏅' + people[0] + ' ' + people[1] + ' ' + people[2] + ' ' +
str(people[3]) + '\n'
    await message.answer('<b>' + ans + '</b>', parse_mode='html')
```

```
async def echo_send_cbq(callback: types.CallbackQuery):
    await callback.delete()


def register_handlers_admin(dp: Dispatcher):
    dp.register_message_handler(start, commands=['start'])
    dp.register_message_handler(help, commands=['help'])
    dp.register_message_handler(game, commands=['game'], state=None)
    dp.register_callback_query_handler(guess_cbq, Text(equals='guess'),
state=FSMGame.game)
    dp.register_callback_query_handler(prompt_cbq, Text(equals='prompt'),
state=FSMGame.game)
    dp.register_callback_query_handler(nat_cbq, Text(equals='nation'),
state=FSMGame.prompt)
    dp.register_callback_query_handler(pos_cbq, Text(equals='position'),
state=FSMGame.prompt)
    dp.register_callback_query_handler(club_cbq, Text(equals='club'),
state=FSMGame.prompt)
    dp.register_callback_query_handler(rate_cbq, Text(equals='skills'),
state=FSMGame.prompt)
    dp.register_callback_query_handler(back_cbq, Text(equals='back'),
state=FSMGame.prompt)
    dp.register_callback_query_handler(true_cbq, Text(equals='1'),
state=FSMGame.guess)
    dp.register_callback_query_handler(false_cbq, Text(equals='2'),
state=FSMGame.guess)
    dp.register_callback_query_handler(false_cbq, Text(equals='3'),
state=FSMGame.guess)
```

```
    dp.register_callback_query_handler(false_cbq, Text(equals='4'),
state=FSMGame.guess)
    dp.register_callback_query_handler(back_cbq, Text(equals='back'),
state=FSMGame.guess)
    dp.register_message_handler(record, commands=['record'])
    dp.register_message_handler(echo_send_cbq, state=FSMGame.game)
    dp.register_message_handler(echo_send_cbq, state=FSMGame.prompt)
    dp.register_message_handler(echo_send_cbq, state=FSMGame.guess)
```

Додаток В.7

Лістинг модуля для створення клавіатури та кнопок ігрового процесу

```
#game_keyBoard/game_KB.py
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton


ilkb = InlineKeyboardMarkup(row_width=1)
promptBut = InlineKeyboardButton(text='🆘 Підказки', callback_data='prompt')
guessBut = InlineKeyboardButton(text='🕵 Вгадати', callback_data='guess')
ilkb.add(promptBut, guessBut)

pkb = InlineKeyboardMarkup(row_width=2)
p1 = InlineKeyboardButton(text='🇺🇳 Нація', callback_data='nation')
p2 = InlineKeyboardButton(text='⚽ Клуб', callback_data='club')
p3 = InlineKeyboardButton(text='🏟 Позиція', callback_data='position')
p4 = InlineKeyboardButton(text='⭐ Рейтинг', callback_data='skills')
back = InlineKeyboardButton(text='◀ Назад', callback_data='back')
pkb.add(p1, p2, p3, p4, back)
```

Додаток В.8

Лістинг файлів для створення пакетів та конфігураційного файлу

```
#handlers/__init__.py
from handlers import admin
from handlers import users


#data_base/__init__.py
from data_base import sqlite_db


#option.py
TOKEN = " 5374224597:AAHJjv1whBxV8Z2tH6t6_w_djByc5j33Jz0"
ADMIN_ID = 375481156
```

Додаток В.9

Лістинг тестів алгоритму веб-скрапінгу

```
import unittest
from cards.parse_cards import parse


class TestParse(unittest.TestCase):
    def test_values(self):
        self.assertRaises(ValueError, parse, -5)
        self.assertRaises(ValueError, parse, -1)

    def test_types(self):
        self.assertRaises(TypeError, parse, 5 + 2j)
        self.assertRaises(TypeError, parse, 'five')
        self.assertRaises(TypeError, parse, [1, 2])
        self.assertRaises(TypeError, parse, [12])
        self.assertRaises(TypeError, parse, True)
```

Додаток В.10

Лістинг тестів алгоритму рандомайзера

```
import unittest
from handlers.users import randUnique4num


class TestRandUnique4num(unittest.TestCase):
    def test_Return(self):
        self.assertEqual(type(randUnique4num(1, 4)), list)

    def test_values(self):
        self.assertRaises(ValueError, randUnique4num, 1, -4)
        self.assertRaises(ValueError, randUnique4num, 1, 0)

    def test_types(self):
        self.assertRaises(TypeError, randUnique4num, 1, 2j)
        self.assertRaises(TypeError, randUnique4num, 1, 'five')
        self.assertRaises(TypeError, randUnique4num, 1, [1, 2])
        self.assertRaises(TypeError, randUnique4num, 1, [12])
        self.assertRaises(TypeError, randUnique4num, 1, False)
```