НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислюваної техніки

До захисту допущено:
В.о завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
(підпис)
“___” _____ 2025р

Звіт з практики

на здобуття ступеня бакалавра за освітньо-професійною програмою
“Інженерія програмного забезпечення комп’ютерних систем”
спеціальності 121 “Інженерія програмного забезпечення”

на тему: Інтелектуальна система для відслідковування спортивних досягнень працівників компанії

Виконав: студент 4 курсу, групи ІМ-13
(шифр групи)

Потапенко Михайло Віталійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент, доктор філософії Шульга М.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ас. Пономаренко А. М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент асистент кафедри БМК Матвійчук О.В _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислюваної техніки

ЗАТВЕРДЖУЮ

В.о завідувача кафедри

_____ Михайло НОВОТАРСЬКИЙ

(підпис)

“___” _____ 2025р

ЗАВДАННЯ

на бакалаврський дипломний проєкт студента

Потапенка Михайла Віталійовича

1. Тема проєкту *Інтелектуальна система для відслідковування спортивних досягнень працівників компанії*
Керівник проєкту асистент, доктор філософії Шульга М.В.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 23 травня 2025 року № 1705-с

2. Термін здачі студентом закінченого проєкту *31 травня 2025 р.*

3. Вихідні дані до проєкту *технічне завдання*

4. Зміст пояснювальної записки (перелік розроблюваних питань)
Розділ 1: Аналіз предметної області
Розділ 2: Огляд підходів та технологій для розробки інтелектуальної системи
Розділ 3: Проектування та реалізація інтелектуальної системи
Розділ 4: Тестування та оцінка працездатності інтелектуальної системи

5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень) структурна схема системи, узагальнена схема роботи системи, схема алгоритму модуля редактора, схема _____ інформаційних вікон редактора.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормконтроль	Пономаренко А. М.		

7. Дата видачі завдання_____

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*09.01.2025-17.01.2025*	
2.	*Вивчення та аналіз завдання*	*22.01.2025-06.04.2025*	
3.	*Розробка архітектури та загальної структури системи*	*11.03.2025-27.03.2025*	
4.	*Розробка структур окремих підсистем*	*28.03.2025-08.04.2025*	
5.	*Програмна реалізація системи*	*08.04.2025-13.05.2025*	
6.	*Оформлення пояснювальної записки*	*14.05.2025-31.05.2025*	
7.	*Тестування, доопрацювання*	*06.06.2025-08.06.2025*	
8.	*Передзахист*	*11.06.2025*	
9.	*Захист*	*19.06.2025*	

Студент-дипломник _____ Михайло ПОТАПЕНКО
(підпис)

Керівник проєкту _____ Максим ШУЛЬГА
(підпис)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів – загалом 73 сторінки. У проєкті було досліджено тему інтелектуальних систем, їх взаємодію з різними базами даних та розроблено інтелектуальну систему для відслідковування спортивних досягнень працівників компанії. В створеній системі є 2 вида користувачів: клієнти – мають зареєструватись та запустити підтвердження атлетичних досягнень, кількість клієнтів необмежена та може сягати кількох тисяч та адміністратори, які через певні інтервали часу заходять в іншу інтелектуальну систему (напряму пов’язану з клієнтською, проте доступ до неї мають лише адміністратори) та витягують звідти інформацію, на основі якої створюють звіт для керівництва на основі даної інформаційної системи (можна по видам спорту, по регіонам, по прізвищам, по досягненням).

Інтелектуальна система напряму пов’язана з базою даних PostgreSQL.

ANNOTATION

The explanatory note of the diploma project consists of four sections - a total of 73 pages. The project investigated the topic of intelligent systems, their interaction with various databases and developed an intelligent system for tracking the sports achievements of company employees. There are 2 types of users in the created system: clients - they must register and run confirmation of athletic achievements, the number of clients is unlimited and can reach several thousand, and administrators, who at certain intervals of time enter another intelligent system (directly connected to the client system, but only administrators have access to it) and extract information from there, on the basis of which they create a report for management based on this information system (you can by sports, by regions, by surnames, by achievements).

The intelligent system is directly connected to the PostgreSQL database.

Довідки	Формат	Значення	Найменування	Кіл. аркушів	№ екземпляра	Додаток
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Інтелектуальна система для	3		
			відслідковування спортивних			
			досягнень працівників компанії			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Інтелектуальна система для	73		
			відслідковування спортивних			
			досягнень працівників компанії			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Інтелектуальна система для	1		
			відслідковування спортивних			
			досягнень працівників компанії			
			Алгоритм роботи системи			

Зм.	Арк	№ докум.	Підпис	Дата	***ІАЛЦ.467200.001 ОА***		
Розроб.		Потапенко М.В.			*Інтелектуальна система для відслідковування спортивних досягнень працівників компанії* Опис Альбому	*Літ.*	*Арк.* 1 / *Аркушів* 2
Перевір.		Шульга М.В.					
Н.контр		Пономаренко А.М				***КПІ ім. Ігоря Сікорського ФІОТ, ІМ-13***	
Затвер							

Довідки	Формат	Значення	Найменування	Кіл. аркушів	№ екземпляра	Додаток
	А4	***ІАЛЦ.467200.005 Д2***	Інтелектуальна система для	1		
			відслідковування спортивних			
			досягнень працівників компанії			
			Загальна структура побудови коду			
	А4	***ІАЛЦ.467200.006 Д3***	Інтелектуальна система для	1		
			відслідковування спортивних			
			досягнень працівників компанії			
			Структурна схема бази даних			
	А4	***ІАЛЦ.467200.007 Д4***	Інтелектуальна система для	16		
			відслідковування спортивних			
			досягнень працівників компанії			
			Текст програмного коду			

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Інтелектуальна система для відслідковування спортивних досягнень працівників компанії*»

ІАЛЦ.467200.002 ТЗ

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

Це технічне завдання поширюється на розробку програмного продукту на тему: «Інтелектуальна система для відслідковування спортивних досягнень працівників компанії». Область застосування: заохочення до спорту співробітників різноманітних компаній.

2. ПРИЧИНИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського проєкту за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем» спеціальності «121 Інженерія програмного забезпечення».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проєкту є розробка інтелектуальної системи для відслідковування спортивних досягнень працівників компанії.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з комп'ютерних технологій, баз даних, публікації в періодичних виданнях, довідники на платформах дистанційного навчання, публікації в Інтернеті з цих питань.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до розробленого продукту:

- Зрозумілий адаптивний інтерфейс інтелектуальної системи
- Швидка обробка команд користувача
- Чіткі та зрозумілі команди для користувачів
- Підтримка на різних платформах

5.2 Вимоги до програмного забезпечення:

Для серверу:

- операційна система MS Windows 10/8/7/Vista/2003/XP або Ubuntu;
- Python 2.4 або вище, PyPy або IronPython

Для користувача:

- встановлений застосунок Telegram

- операційна система MS Windows 10/8/7/Vista/2003/XP;

5.3 Вимоги до апаратної частини:

- комп’ютер на базі процесора Intel Pentium 166 і вище
- оперативна пам’ять не менше 2Гбайт
- мінімальна роздільна здатність екрану 1024x768
- вільний простір жорсткого диску не менше 300 Мбайт підключення до Інтернету

6. ЕТАПИ РОЗРОБКИ

Назва етапів розробки	Термін виконання
Затвердження теми роботи	09.01.2025-17.01.2025
Вивчення та аналіз завдання	22.01.2025-06.04.2025
Розробка архутектури та загальної структури системи	11.03.2025-27.03.2025
Розробка структур окремих частин системи	28.03.2025-08.04.2025
Програмна реалізація системи	08.04.2025-13.05.2025
Виправлення помилок	13.05.2025-14.05.2025
Оформлення пояснювальної записки	14.05.2025-31.05.2025

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Інтелектуальна система для відслідковування спортивних досягнень працівників компанії*»

ІАЛЦ.467200.003 ПЗ

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

ШІ – Штучний Інтелект

NLP – Natural Language Processing

XML – eXtensible M Language

JSON – JavaScript Object Notation

IBM – International Business Machines

BRMS – Business Rules Management System

SQL – Structured Query Language

XAI – eXplainable Artifical Intelligence

API – Application Programming Interface

UML – Unified Modeling Language

LLMs –Large Language Models

HTTP – HyperText Transfer Protocol

ML – Machine Learning

ACID – Atomicity, Consistency, Isolation, Durability

IDE – Integrated Development Environment

ORM – Object-Relational Mapping

GIT – Global Information Tracker

VCS – Version Control System

ID – IDentifier

FSM – Finite State Machine

FTP – File Transfer Protocol

VPS – Virtual Private Server

VDS – Virtual Dedicated Server

RegExp – Regular Expressions

SSH – Secure SHell

ПЕОМ – Персональна Електронно-Обчислювальна Машина

ВСТУП

У сучасному світі, де динамічний розвиток технологій стає рушійною силою прогресу, інтелектуальні системи виходять на передовий план, трансформуючи підхід до взаємодії, аналізу та вирішення складних завдань. Ця робота занурюється у світ штучного інтелекту, досліджуючи його історичний шлях від філософських концепцій до епохи глибоких нейронних мереж і великих мовних моделей, що нині формують ландшафт інновацій. Буде детально розглянуто не лише еволюцію цієї захопливої галузі, а й специфіку її застосування в сучасних умовах, зосередившись на розробці інтелектуальної системи, покликаної об'єднати та мотивувати співробітників компаній через спортивну активність. Особливу увагу буде приділено аналізу предметної області, принципам проектування та реалізації, а також всебічному тестуванню системи, що демонструє її функціональність, надійність та готовність до ефективного використання у реальному світі. Цей проєкт – це не просто технологічне рішення, а крок до створення більш інтегрованого, здорового та продуктивного корпоративного середовища.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Історія та розвиток інтелектуальних систем

Історія та розвиток інтелектуальних систем — це захоплива подорож від теоретичних концепцій до складних технологій, які можна побачити сьогодні. Цей багатогранний процес охоплює десятиліття наукових відкриттів, технологічних проривів та змін у підходах до створення машин, здатних імітувати або перевершувати людський інтелект.

1.1.1 Перші ідеї та кроки в ШІ

Сама ідея створення ШІ сягає глибокої давнини, з міфами про штучних істот та філософськими роздумами про природу розуму. Грецькі філософи заклали основи формальної логіки, яка згодом стала важливим інструментом для розвитку ШІ. У XVII-XIX століттях математики розробили формальні системи для представлення та маніпулювання знаннями. Булева алгебра, зокрема, стала фундаментальною для розвитку комп'ютерів та, відповідно, ШІ.

Також важливо згадати про аналітичну машину Беббіджа та алгоритми Лавлейс. Хоча це не був ШІ в сучасному розумінні, проєкт Чарльза Беббіджа щодо аналітичної машини і перші алгоритми, написані Адою Лавлейс для цієї машини, заклали теоретичну основу для програмованих обчислень.

1.1.2 Золота ера ШІ

В 1950-х роках починається, так звана, золота ера ШІ.

Дартмутський семінар в 1956 році вважається офіційним народженням штучного інтелекту як наукової дисципліни. Було організовано семінар, на

якому проголосили, що "будь-який аспект навчання або будь-яка інша особливість інтелекту в принципі може бути настільки точно описана, що машину можна буде змусити її імітувати".

У цей період було розроблено перші програми, які демонстрували елементи "інтелектуальної" поведінки: Logic Theorist - програма, яка могла доводити математичні теореми, General Problem Solver (GPS) - амбітна спроба створити універсальну програму для розв'язання проблем, ELIZA - програма-чатбот, яка імітувала розмову з психотерапевтом та SHRDLU - програма, яка розуміла обмежену природну мову в контексті блокового світу.

У цей час активно розвивалися теорії символьного ШІ, засновані на представленні знань у вигляді символів та правил. Перші комп’ютери лише починали обчислення, але ідеї штучного інтелекту вже з’явилися.

Основними досягненнями золотої ери було винайдення першої програми, що вирішувала завдання загального характеру в 1959 році - **General Problem Solver.** Активно розвивалась розробка систем, які імітували рішення експертів у вузьких сферах (наприклад, медицина, діагностика). Наприклад, програма, що імітувала психотерапевта, спілкуючись із користувачем - **Eliza (1966)**

1.1.3 Зима ШІ

Наступний період у розвитку штучного інтелекту припав на 1970-ті роки та початок 1980-х років. Він отримав назву «Зима ШІ» через певне розчарування та спад фінансування даної галузі.

Перші успіхи породили завищені очікування щодо швидкого створення "справжнього" інтелекту. Однак складність реальних проблем виявилася значно більшою, ніж передбачалося.

Виникали проблеми з масштабуванням - програми, які добре працювали в обмежених доменах, часто не могли впоратися зі складнішими та реалістичнішими сценаріями.

Доступні на той час комп'ютери мали обмежену обчислювальну потужність та пам'ять, що ускладнювало розробку складних ШІ-систем. В 1973 році з’явився звіт Lighthill - критичний звіт британського уряду, який вказував на відсутність значного прогресу в ШІ та призвів до скорочення фінансування досліджень у Великій Британії. Подібні тенденції спостерігалися й в інших країнах.

В зв’язку з обмеженістю обчислювальних ресурсів та неможливістю реалізувати "загальний інтелект" сфера отримала наслідки у вигляді зменшення фінансування досліджень та переорієнтації на більш вузькі та прикладні системи.

1.1.4 Відродження та експертні системи

В середині 1980-х роках почалось відродження галузі та здобуті знання почали застосовуватись практично.

Цей період ознаменувався комерційним успіхом різноманітних експертних систем – програм, які використовували набори правил ("if – else") для розв'язання проблем у вузьких областях знань. Системи, що використовують базу знань і логічні правила для прийняття рішень. Вперше інтелектуальні системи почали застосовувати в бізнесі: прогнозування, планування, технічна підтримка.

В той же час починається розвиток логічного програмування: Мова програмування Prolog, заснована на принципах логіки, стала популярною для розробки ШІ-систем.

1.1.5 Друга зима ШІ

На початкових етапах розвитку штучного інтелекту не було стабільності. Стрімкі підйоми чергувались з різкими спадами. Так, наприкінці 1980-х років почалась друга зима ШІ. Розробники розчарувались в експертних системах.

Експертні системи виявилися складними в розробці та підтримці, особливо при розширенні області їх застосування. Вони також мали труднощі з обробкою невизначеності та навчанням на нових даних. Інтерес та фінансування досліджень в області ШІ знову пішли на певний спад.

Першою ознакою “похолодання” був неочікуваний крах ринку спеціалізованого апаратного забезпечення для ШІ в 1987 році. Зрештою, найперші успішні експертні системи стали дуже дорогими в обслуговуванні, ускладнилась процедура їх оновлення. Понад 300 компаній, які спеціалізувались на штучному інтелекті зникли через банкрутство, тим самим де-факто завершивши першу комерційну хвилю ШІ. Наприкінці 1980-х років кілька дослідників активно просували ідею щодо нового підходу до ШІ на основі робототехніки. Цей підхід відродив ідеї з теорії керування та кібернетики, які були популярні з 60-х років XX століття.

Загалом важливо зауважити, що кожен період приносив у програмування штучного інтелекту щось нове, приносив новий досвіт, який в майбутньому був сповна використаний. Завдяки покращенню технологій, збільшувалась і кількість ідей та робота дослідників.

1.1.6 Епоха машинного навчання

Станом на 2025 рік триває, так звана, епоха машинного навчання. Розпочалась ця епоха в середині 1990-х років. Після періоду відносної забуття, нейронні мережі знову привернули увагу завдяки прогресу в обчислювальній потужності та наявності великих обсягів даних. Розвиток статистичних методів машинного навчання, таких як опорні векторні машини (SVM), випадкові ліси (Random Forests) та градієнтний бустинг, призвів до значних успіхів у різних областях.

Починають використовуватися **методи машинного навчання** замість жорстко заданих правил та досягаються перші успіхи в комп’ютерному баченні та обробці мов програмування. В якості прикладів можна навести

Deep Blue — комп’ютер IBM, який переміг чемпіона світу Гаррі Каспарова у шахах та автономну машину **ALVINN**, що керувала автомобілем на шосе.

Швидко почали з’являтися потужні обчислювальні ресурси (GPU) та великі набори даних. Їх поява дала змогу тренувати глибокі нейронні мережі з багатьма шарами, що призвело до революційних проривів у таких областях, як розпізнавання образів, обробка природної мови та машинний переклад. Доступність величезних обсягів даних стала ключовим фактором успіху багатьох алгоритмів машинного навчання.

Сьогодні інтелектуальні системи широко використовуються в багатьох галузях, включаючи медицину, фінанси, транспорт, освіту, розваги та багато інших. Ми бачимо їх у вигляді віртуальних помічників, систем рекомендацій, автономних транспортних засобів, систем розпізнавання облич та голосу тощо.

Інтелектуальні системи сьогодні застосовують в медицині, в фінансах, в автомобілях та в креативності та мистецтві.

1.1.7 Проблематика

Незважаючи на стрімкий розвиток штучного інтелекту, інформаційних систем та галузі інформаційних технологій загалом, поки що є багато програм, які працюють з помилками або виконують не всі потрібні функції, які мали б виконувати застосунки подібного спрямування. Багаторазово було помічено цю проблему і розглянувши один із можливих сценаріїв, було прийнято рішення щодо його вдосконалення та створення універсальної інформаційної системи.

В світі існує багато великих корпорацій, компаній та фірм які намагаються взаємодіяти зі своїми співробітниками. Одним із способів побудови хорошої комунікації є система премій та нагород, які мотивують працівників виконувати роботу з потрібною продуктивністю та якістю. Водночас для побудови хороших взаємовідносин між керівниками та підлеглими, час від часу потрібно влаштовувати процеси, які об’єднують всіх в одне ціле та

створюють з абсолютно різних та чужих один одному людей справжню команду! Часто, так званий тімбілдінг, роблять за допомогою поїздки на спільний відпочинок або завдяки занять спортом. Зупинимось на 2 варіанті.

Спорт – одне з найбільш популярних та корисних занять в житті кожної людини. Багато компаній, окрім свого профільного спрямування, намагаються заохочувати своїх співробітників до здорового способу життя і в такій ситуації саме спорт є одним із головних аспектом, який може вплинути на здоров'я та загальний фізичний стан людини.

Декотрі керівники готові надавати працівникам своєї фірми можливість та всі умови для занять спортом. Зокрема, оплачувати абонементи в тренажерний зал, басейн та інші заклади з спортивним спрямуванням. Проте, абсолютно всі керівники та бізнесмени рахують свої гроші і не мають бажання вкладати фінанси в те, що не принесе користі. Тому їм важливо, щоб співробітник не просто числився як людина якій потрібно займатися спортом, а ще й займався ним.

Для цих потреб спершу було створено реєстр, в який один із співробітників записував всю інформацію про спортивні досягнення своїх колег. Записувався вид спорту та час занять для кожного працівника. Згодом цю ідею масштабували від розміру одного окремого відділу, до розмірів цілої компанії і ведення звіту одним або кількома працівниками компанії, в зв'язку з цим стало фізично неможливим через дуже велику кількість людей та інформації, яку потрібно було систематизувати в одному місці. Потрібно було зробити машинальне заповнення списків всіх співробітників, які займаються спортом, для ведення статистики.

Компаніям, які хочуть заохочувати своїх співробітників до спортивної діяльності, потрібна якісна та проста у використанні інформаційна система, щоб за допомогою якоїсь бази даних накопичувати інформацію про всі досягнення, яку можна буде фільтрувати по філіалах, по регіонах, по виду спорту та інших критеріях тощо.

Проте із всіх існуючих програм, в яких хоча б частково створена система для заповнення спортивних досягнень, немає жодної досконалої, в якій можна було б виконувати всі ті функції, які були наведені вище. Тому було прийнято рішення створити власну інформаційну систему, яка буде поєднувати в собі всі кращі сторони вже існуючих застосунків, таким чином утворюючи ідеальну систему, якою будуть користуватись всі охочі.

1.1.8 Сучасні тенденції та майбутнє

Зараз ШІ розвивається дуже швидко. Глибокі нейронні мережі дали змогу досягти прориву в розпізнаванні зображень (ImageNet), обробці мови (Google Translate, Siri) та в генерації текстів.

Зростає потреба в розумінні того, як ШІ-системи приймають рішення, особливо у критично важливих областях. Прогресуватиме подальший розвиток систем, здатних розуміти, інтерпретувати та генерувати людську мову. Також варто очікувати удосконалення систем, здатних "бачити" та інтерпретувати візуальну інформацію та покращення інтеграція ШІ в системах з роботами для створення більш інтелектуальних та автономних пристроїв.

Все більше буде реалізовуватися довгострокова мета створення ШІ, здатного до розуміння, навчання та застосування знань у широкому спектрі завдань на рівні або вище людського.

Історія розвитку інтелектуальних систем – це історія амбіцій, розчарувань та яскравих проривів. Нинішнє покоління є свідками нової хвилі інновацій, яка змінює наш світ. І хоча шлях до "справжнього" ШІ все ще може бути довгим, досягнутий прогрес є вражаючим і обіцяє нові відкриття у майбутньому.

1.2 Огляд існуючих рішень

Перед тим як розглядати існуючі програми для створення інтелектуальної системи для заповнення спортивних результатів, потрібно детально розібрати

існуючі рішення та системи в аналізі предметної області в інтелектуальних системах. Ця галузь є надзвичайно широкою та охоплює різноманітні підходи, методи та інструменти, спрямовані на розуміння, представлення та використання знань про конкретну сферу діяльності.

1.2.1 Основні цілі аналізу предметної області

Аналіз предметної області дуже важливий перед виконанням будь-якої задачі. Важливо розуміти, які існують рішення для виконання тої чи іншої задачі. Основні цілі аналізу предметної області в інтелектуальних системах:

- Виявлення та формалізація знань: Отримання експертних знань, даних та інформації про предметну область та їх представлення у формальному вигляді, зрозумілому для комп'ютерних систем.
- Побудова онтологій: Створення структурованих моделей знань, що визначають концепції предметної області, їхні властивості та взаємозв'язки.
- Семантична інтеграція даних: Об'єднання даних з різних джерел на основі їхнього семантичного значення, що дозволяє здійснювати більш глибокий аналіз та виводи.
- Розробка інтелектуальних застосунків: Створення систем, які можуть розуміти контекст, робити висновки, приймати рішення та взаємодіяти з користувачем на основі знань про предметну область.

1.2.2 Сучасні тенденції

Інтеграція машинного навчання та символьних методів створюється за допомогою розробки гібридних систем, які поєднують вміння та здатності машинного навчання до обробки великих обсягів даних з різними логічними висновками та структурованістю онтологій.

В наш час використовуються здебільшого великі мовні моделі. Зокрема для автоматичного вилучення знань, побудови онтологій та розробки

інтелектуальних застосунків. Водночас із цим створюються нові системи, які мають вбудовану здатність до врахування контексту при аналізі предметної області.Відбувається потужний розвиток контекстно-залежного розуміння.

З кожним роком ця галузь стає все більш масштабованою та ефективною, оскільки щорічно розробляють багато ефективних та масштабованих систем для обробки великих обсягів даних та знань. Важливо зауважити, що створюються стандарти та інструменти для забезпечення сумісності та обміну знаннями між різними системами.

Аналіз предметної області є ключовим етапом у розробці інтелектуальних систем. Існує широкий спектр різноманітних рішень та систем, які базуються на різних підходах до представлення знань, типах аналізованих даних та рівнях автоматизації. Сучасні тенденції спрямовані на інтеграцію різних методів та використання новітніх досягнень у галузі штучного інтелекту для створення більш потужних, гнучких та ефективних інтелектуальних систем.

1.2.3 Існуючі програми для фіксації спортивних результатів

Підтримувати мотивацію співробітників можна завдяки застосункам для бігу. Всі вони не ідеальні, але також варті уваги:

1. «WOWBODY» – українська програма, яка містить різні програми тренувань. В програмі є безкоштовний пробний період, який триває протягом тижня. Проте після цього пробного періода не буде жодних пробних функцій. Застосунок підтримує різний формат тренувань та враховує фізичні можливості.

2. «Adidas Runtastic: Run Tracker» – застосунок пропонує додати цілі щодо дистанції, часу та кількості тренувань. Великим мінусом є відсутність української мови. Серед видів активності є їзда на лижах, біг, велоїзда, верхова їзда, веслування, ходьба, їзда на велотренажері, плавання, ходіння сходами тощо.

3. «Caynax Sports Tracker» – програма для поціновувачів мінімалізму. Має доступні функції бігу, веслування тощо. Працює лише на Android та має небагато функцій. Немає спільнот та персоналізованих функцій.

4. «ASICS Runkeeper — Run Tracker» – непоганий застосунок для спорту, має багато функцій, проте багато з них недоступні без платної підписки, а також в програмі відсутня українська мова, що може створити дискомфорт з перекладом.

5. «Strava» – один із найбільш відомих застосунків для бігу, яким користується багато людей по всьому світу. Серед проблем відсутність української мови інтерфейсу та відсутність мапи. Strava є одним з найкращих застосунків для заповнення спортивних досягнень.

Незважаючи на плюси різних застосунків, в кожному з них є мінуси. В одних відсутня українська мова, в інших мало безкоштовних функцій. Тому є необхідність створення якісного застосунку, який буде в собі поєднувати більшість плюсів та, по можливості, уникати мінуси існуючих сервісів.

1.3 Визначення вимог до нової інтелектуальної системи

Процес створення інтелектуальної системи є складним та багатоетапним завданням, що потребує чіткого визначення функціональних та нефункціональних вимог. Від якості стартового етапу – формулювання вимог до системи – залежить не лише ефективність роботи системи, але й її життєздатність, масштабованість та здатність до адаптації.

1.3.1 Загальні принципи визначення вимог до інтелектуальної системи

Перед розробкою будь-якої інтелектуальної системи необхідно провести глибокий аналіз проблемної області. В цьому аспекті найважливішим є кілька факторів, на які важливо звернути увагу. Серед них вивчення контексту, в

якому функціонуватиме система, ідентифікація нових учасників та опис бізнес-процесів, які система має автоматизувати чи підтримувати.

Аналіз проблемної області дозволяє сформулювати чітке бачення задач, які стоятимуть перед новоствореною інтелектуальною системою. На цьому етапі збираються вимоги зацікавлених сторін, що дає змогу уникнути непорозумінь та суперечок у подальших фазах розробки.

1.3.2 Вимоги

- **Функціональні** вимоги описують, що саме має робити система. В інтелектуальних системах до функціональних вимог належать: обробка природної мови (NLP), класифікація даних, прийняття рішень на основі експертних правил чи машинного навчання, адаптивність до змінних умов середовища та ведення діалогу з користувачем (у випадку чат-ботів). Ці вимоги повинні бути формалізовані у вигляді конкретних сценаріїв використання (use cases) або у вигляді діаграм UML (діаграми прецедентів, послідовностей тощо).
- **Нефункціональні** вимоги охоплюють ті характеристики системи, які не пов’язані безпосередньо з її поведінкою, але критично впливають на якість продукту. До таких вимог належить продуктивність (response time, обробка великої кількості запитів), можливість обробляти збільшену кількість даних або користувачів (масштабованість), надійність та стійкість до помилок, безпека у вигляді якісного захисту особових даних (використовується авторизація), доступність (важливо, аби система працювала цілодобово та підтримувалась на різних платформах), а також зручність у використанні.
- **Архітектурні** та **технологічні** вимоги – це вимоги до середовища розгортання. Наприклад, хмарне сховище або локальний сервер. Також саме тут визначаються вимоги до мови програмування, баз даних, фреймворків та інших компонентів Окрім того, слід також враховувати можливість інтеграції

з іншими системами за допомогою API, Webhook або іншими протоколами обміну даними.

1.3.3 Telegram-бот як приклад інтелектуальної системи

Telegram-боти – це дуже комфортний гнучкий інструмент для створення інтелектуальних систем, що дуже допомагає в створенні та дозволяє швидко реалізувати систему взаємодії з користувачем через простий текстовий інтерфейс.

Популярність Telegram-ботів зумовлена кількома перевагами:

- Простота інтеграції через Bot API від Telegram
- Підтримка широкого спектра різноманітних повідомлень, таких як текст, зображення, кнопки, меню тощо
- Висока доступність: для користування ботом не потрібно встановлювати додаткові програми
- Підтримка webhooks, що дозволяє швидко реагувати на повідомлення
- Безкоштовний хостинг з боку Telegram
- Зручна авторизація та ідентифікація користувача
- Вбудовані інструменти безпеки

Водночас із тим, бот, має виконувати вимоги, як функціональні, так і нефункціональні.

Серед функціональних вимог можна виділити розпізнавання запитів користувача, коли за допомогою алгоритмів обробки природної мови бот має розуміти наміри користувача та відповідно обирати реакцію на них. Бот повинен вміти вести контекстний діалог – зберігати стан діалогу та реагувати з урахуванням попередніх повідомлень, що дозволяє робити взаємодію більш природною. Окрім того, має бути здатність автоматичного надання інформації. Наприклад, бот може бути використаний для надання консультацій, пошуку інформації в базі знань або генерації текстів. Важливою рисою інтелектуальної системи є здатність до самонавчання. Бот має вміти адаптувати свої відповіді

на основі зворотного зв’язку від користувачів. Ну і насамкінець, кожен бот повинен мати доступ до джерел інформації, зокрема баз даних, API зовнішніх сервісів або моделей машинного навчання.

Щодо нефункціональних вимог до Telegram-бота, то кожен бот повинен бути швидким та відповідати протягом секунд (1-2) на звичайні запити. У разі зростання аудиторії система має легко масштабуватися. Також повинна бути надійність та безпека. Має бути передбачена авторизація, захист персональних даних та обмеження доступу користувачів до адміністративного функціоналу. Бот має мати логічну структуру меню, кнопки, інтерактивні елементи, які підвищують зручність користувача. Будь-який бот в Telegram повинен працювати цілодобово (24/7) без збоїв та на випадок відмови якихось сервісів, має застосовуватись резервне копіювання, аби дані не були втрачені.

1.3.4 Особливості визначення вимог

Для визначення вимог, можуть використовуватись різні методи. Зокрема, аналіз існуючих рішень – огляд конкурентів або аналогічних систем. Це допоможе уникнути типових помилок. Також можу порадити провести інтерв’ю зі стейкхолдерами. Пряма комунікація допоможе виявити очікування користувачів. Для якісного визначення вимог слід застосувати прототипування, що несе під собою створення швидкого прототипу (MVP) для демонстрації ключових функцій збору зворотного зв’язку.

Правильне визначення вимог до інтелектуальної системи – це фундамент, на якому базується вся розробка. Не менш важливо враховувати як функціональні, так і нефункціональні вимоги, а також технологічні обмеження й очікування користувачів.

Telegram-бот є прикладом інтелектуального агента, демонструє простоту реалізації, ефективність і гнучкість, завдяки чому є ідеальним вибором для розробки сучасної системи взаємодії.

Висновки до розділу 1

Аналіз історії розвитку інтелектуальних систем дає змогу зробити висновок, що ця галузь є однією з тих, які розвиваються найбільш динамічно і однією з найперспективніших у сучасній науці та техніці. Її формування проходило через низку етапів, кожен з яких відіграв ключову роль у становленні сучасних підходів до створення штучного інтелекту.

Еволюція інтелектуальних систем — це безперервний процес удосконалення технологій, методів і підходів, що має глибоке наукове та практичне значення для подальшого розвитку сучасного суспільства.

Проведений огляд існуючих рішень у сфері аналізу предметної області в інтелектуальних системах засвідчив багатство та різноманітність підходів до представлення та обробки знань. Ефективна робота інтелектуальних систем значною мірою залежить від якості аналізу предметної області. Аналіз предметної області є не лише початковим, а й стратегічно важливим етапом у розробці інтелектуальних систем. Його якість визначає ефективність, точність і здатність системи адаптуватися до реальних умов.

Важливу роль у побудові систем відіграє тип даних, що підлягають аналізу. Системи можуть бути орієнтовані на структуровані, неструктуровані або напівструктуровані дані, що вимагає відповідного вибору інструментів та технологій.

Особливу увагу варто приділити прикладу Telegram-бота як однієї з найзручніших і найефективніших форм інтелектуального агента, здатного забезпечити зручну взаємодію з користувачем, швидку обробку запитів, інтеграцію з зовнішніми сервісами та підтримку самонавчання.

Правильно сформульовані вимоги до інтелектуальної системи є основою для подальшої якісної розробки. Вони дозволяють забезпечити не лише відповідність очікуванням користувачів, а й технологічну життєздатність системи у довгостроковій перспективі.

РОЗДІЛ 2
ОГЛЯД ПІДХОДІВ ТА ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ

2.1 Порівняльний аналіз існуючих рішень

У сучасному цифровому світі Telegram-боти стали потужним інструментом для автоматизації різноманітних завдань, починаючи від надання інформації та підтримки клієнтів до керування бізнес-процесами. Їхня популярність зумовлена зручністю використання, широким спектром функціональних можливостей та інтеграцією з однією з найпопулярніших платформ обміну повідомленнями. Розробка інтелектуальних телеграм-ботів, здатних розуміти природну мову, навчатися та адаптуватися до потреб користувачів, є актуальним напрямком досліджень та розробок у сфері штучного інтелекту.

2.1.1 Основні підходи до розробки

Розробка інтелектуальних систем може здійснюватися з використанням різноманітних підходів, які можуть відрізнятися один від одного рівнем складності, функціональністю та необхідними знаннями.

Існує велика кількість фреймворків та бібліотек для різних мов програмування, які значно спрощують процес розробки інтелектуальних систем. Ці інструменти надають готові API для взаємодії з Telegram Bot API, обробки оновлень, керування станами та іншими завданнями. Прикладами популярних фреймворків є python-telegram-bot (Python) та Telegraf (Node.js).

Якщо користувач не має глибоких знань в програмуванні, то в нього буде можливість використовувати численні платформи для створення ботів без коду. Ці платформи дозволяють створювати боти з використанням візуальних інтерфейсів та блокового програмування. Вони часто пропонують готові

інтеграції з різними сервісами, включаючи бази даних. Прикладами таких платформ є ManyChat, Chatfuel та Flow XO.

Також можна створювати проєкти та займатися розробкою з використанням Telegram Bot API без фреймворків: Цей підхід передбачає безпосередню взаємодію з Telegram Bot API через HTTP-запити. Він надає максимальну гнучкість, але вимагає значних зусиль для реалізації базової функціональності, такої як обробка оновлень та керування сесіями. Тому робота з базами даних є оптимальним варіантом.

2.1.2 Критерії класифікації за різними підходами

1. За підходом до представлення знань

Існуючі рішення та системи можна класифікувати за різними критеріями. Одних із них - за підходом до представлення знань. Він поділяється на наступні категорії:

- На основі правил (Rule-based systems):

Знання представляються у вигляді набору правил "якщо-тоді" (if-then), які визначають дії або висновки на основі певних умов. Використовується такий підхід у експертних системах (наприклад, MYCIN для діагностики інфекційних захворювань), а також у системах керування бізнес-правилами (BRMS) та ін.

Основними перевагами є чіткість та прозорість представлення знань, легкість розуміння та модифікації правил. Водночас існує складність масштабування для великих і складних предметних областей, а також труднощі з обробкою невизначеності та неповних знань.

- На основі фреймів (Frame-based systems):

Знання організовуються у вигляді фреймів, які представляють типові об'єкти або ситуації предметної області. Кожен фрейм має набір слотів (атрибутів) зі значеннями та можливими обмеженнями. В якості прикладу

можна навести системи представлення знань у штучному інтелекті (наприклад, ранні системи розуміння природної мови).

Серед переваг можна виділити структуроване представлення знань та можливість визначення типових характеристик об'єктів. Серед недоліків обмежену гнучкість у порівнянні з іншими підходами та складність представлення складних взаємозв'язків.

- На основі семантичних мереж (Semantic networks):

Знання представляються у вигляді графа, де вузли представляють концепції або об'єкти, а ребра – семантичні відношення між ними (наприклад, "є", "має властивість", "пов'язаний з").

В якості переваги варто зазначити наочне представлення взаємозв'язків між концепціями та можливість здійснення семантичного пошуку та виведення. А недоліками є складність формалізації складних знань, а також потенційні проблеми з масштабованістю та обробкою великих обсягів інформації.

- На основі онтологій (Ontology-based systems):

Онтології є формальними, явними специфікаціями концептуалізації. Вони визначають класи, властивості, відношення та аксіоми предметної області. Є багато прикладів цього в різних сферах: Web Ontology Language (OWL), Resource Description Framework (RDF), онтології в медицині (SNOMED CT), біології (Gene Ontology.

Тут варто зазначити високий рівень формалізації та структуризації знань, можливість семантичної інтеграції даних, підтримка логічних виводів та міркувань. Але є нюанси зі складністю розробки та підтримки великих і складних онтологій та необхідністю експертних знань у галузі онтологічного інжинірингу.

- На основі машинного навчання (Machine learning-based systems):

Знання про предметну область витягуються автоматично з даних за допомогою алгоритмів машинного навчання. Прикладами цього є системи

виявлення сутностей, класифікація текстів, аналіз тональності, тематичне моделювання, побудова векторних представлень слів та документів.

Перевагою є автоматичне вилучення знань з великих обсягів даних, здатність обробляти неструктуровані дані та адаптивність до змін у даних. Недоліки можна побачити в залежності від якості та обсягу навчальних даних та потенційних проблемах з узагальненням на нові дані.

- Гібридні підходи:

Комбінування різних методів представлення знань для використання їхніх сильних сторін та компенсації слабких. Наприклад, використання онтологій для структурування знань, отриманих за допомогою машинного навчання. Такі підходи активно використовуються в системах семантичного пошуку, інтелектуальних системах рекомендацій та системах обробки природної мови з використанням онтологій.

2. За типом аналізованих даних

- Системи аналізу структурованих даних:

Аналіз даних, організованих у таблицях, базах даних, електронних таблицях, методами якого є SQL-запити, OLAP (Online Analytical Processing), data mining (кластеризація, класифікація, асоціативні правила).

- Системи аналізу неструктурованих даних:

Аналіз текстів, зображень, аудіо, відео, методами якого є обробка природної мови (NLP), комп'ютерний зір (Computer Vision), розпізнавання мови (Speech Recognition) та методи машинного навчання (глибоке навчання).

- Системи аналізу напівструктурованих даних:

Аналіз даних, які мають певну структуру, але не є повністю формалізованими (наприклад, XML, JSON), з методами у вигляді парсингу, вилучення інформації та перетворення в структурований формат.

3. За рівнем автоматизації процесу аналізу

- Ручний аналіз: Виконання аналізу експертами предметної області без використання автоматизованих інструментів.

- Автоматизований аналіз: Використання програмних засобів та алгоритмів для автоматичного вилучення, обробки та аналізу знань.
- Напівавтоматизований аналіз: Комбінація ручної роботи експертів та автоматизованих інструментів для підвищення ефективності та точності аналізу.

4. За орієнтацією на конкретні завдання

- Системи виявлення сутностей та зв'язків. Вилучення з текстів згадок про сутності (наприклад, імена людей, організації, місця) та визначення зв'язків між ними.
- Системи класифікації та категоризації. Призначення об'єктам або документам певних категорій на основі їхніх характеристик або змісту.
- Системи вилучення ключової інформації. Ідентифікації та вилучення найбільш важливих фактів та деталей з документів або даних.
- Системи відповіді на запитання. Надання відповідей на запитання користувача, використовуючи знання про предметну область.
- Системи рекомендацій. Пропонування користувачам релевантних товарів, послуг або інформації на основі їхніх уподобань та історії взаємодії.
- Системи пошуку інформації на основі семантичного значення запиту, а не лише ключових слів.
- Системи моделювання бізнес-процесів (візуалізація, аналіз та оптимізація)

2.1.3 Технології для розробки інтелектуальних можливостей бота

Для надання ботам інтелектуальних можливостей використовуються різні технології штучного інтелекту (ШІ) та обробки природної мови (NLP):

1. Обробка природної мови (NLP): Технології NLP дозволяють ботам розуміти та інтерпретувати текстові повідомлення користувачів. Це включає в себе розпізнавання намірів користувача, вилучення сутностей, аналіз тональності та генерація природної мови для відповідей. Для реалізації NLP

можуть використовуватися як готові хмарні сервіси, так і open-source бібліотеки.

2. Машинне навчання (ML): Алгоритми машинного навчання можуть бути використані для навчання ботів на великих обсягах даних, що дозволяє їм покращувати свою здатність розуміти запити користувачів, персоналізувати відповіді та передбачати їхні потреби. Для навчання та розгортання моделей машинного навчання можуть використовуватися такі фреймворки, як TensorFlow, PyTorch та scikit-learn, а також хмарні платформи машинного навчання.

3. Бази знань: Для надання ботам контекстуальної інформації та можливості логічного виведення можуть використовуватися бази знань та онтології. Вони представляють собою структуровані сховища інформації про предметну область, що дозволяє ботам відповідати на складні запитання та робити висновки на основі наявних даних.

2.1.4 Наповнення інтелектуальної системи через базу даних

Одним з ключових аспектів розробки інтелектуальних телеграм-ботів є ефективне керування даними та їхнє використання для наповнення бота контентом. Існує кілька підходів до інтеграції телеграм-бота з базами даних.

Одним із них є прямий доступ до бази даних з коду бота. Цей підхід передбачає написання коду, який безпосередньо взаємодіє з базою даних для отримання необхідної інформації та її відображення користувачеві через інтерфейс бота. Серед переваг можу виділити прямий контроль над запитами до бази даних та високу гнучкість у формуванні відповідей. Проте такий підхід потребує значних зусиль для написання та підтримки коду, а також потенційні ризики безпеки при неправильній обробці даних.

Іншим, але також дієвим підходом є використання проміжного API. У цьому підході розробляється окремий BackEnd-сервіс, який відповідає за взаємодію з базою даних та надання даних боту через API. Бот надсилає запити

до API, отримує оброблені дані та відображає їх користувачеві. Такий підхід переважає попередній своєю покращеною безпекою, розділенням логіки роботи з базою даних та логіки бота та можливістю використання різних технологій. Тим не менш, він потребує додаткових зусиль на розробку та підтримку BackEnd-сервісу.

Також можна використовувати платформи для створення ботів з інтеграцією баз даних. Деякі платформи для створення ботів без коду або з мінімальним кодуванням пропонують вбудовані можливості для інтеграції з базами даних. Це може включати візуальні інструменти для налаштування запитів та відображення даних. В цього підходу доволі швидка розробка та він не потребує глибоких знань у програмуванні. Але обмежена гнучкість у порівнянні з програмуванням та залежність від можливостей платформи.

2.1.5 Порівняльний аналіз баз даних

Вибір бази даних є критично важливим рішенням при розробці телеграм-бота, оскільки вона визначає спосіб зберігання, керування та доступу до даних, необхідних для функціонування бота. Різні типи баз даних мають свої особливості, переваги та недоліки, які слід враховувати при виборі оптимального рішення.

Існує кілька основних типів баз даних, які можуть бути використані для розробки телеграм-ботів. У ході роботи було розглянуто найголовніші з них та обрано кращий варіант:

1. **Реляційні бази даних (SQL):**

Ці бази даних організовують дані у вигляді таблиць зі стовпцями та рядками, де існують зв'язки між таблицями встановлюються за допомогою ключів. Вони забезпечують високу цілісність даних, підтримку ACID-транзакцій та потужні можливості для складних запитів за допомогою мови SQL. **Прикладами реляційних баз даних є** PostgreSQL, MySQL, SQLite, тощо.

2. **Нереляційні бази даних (NoSQL):**

Ці бази даних використовують різні моделі зберігання даних, відмінні від реляційної моделі. Вони часто характеризуються високою масштабованістю, гнучкістю схеми та кращою продуктивністю для певних типів даних та робочих навантажень.

- **Документо-орієнтовані –** зберігають дані у вигляді JSON-подібних документів (MongoDB).
- **Ключ-значення –** зберігають дані як пари "ключ-значення" (Redis, Memcached).
- **Стовпцево-орієнтовані – з**берігають дані за стовпцями, а не за рядками (Cassandra).
- **Графові – з**берігають дані у вигляді вузлів та ребер, що представляють сутності та їхні зв'язки (Neo4j).

3. **Вбудовані бази даних:**

Це легкі та самодостатні бази даних, які зберігаються безпосередньо у файлі програми. Наприклад, вбудованою базою даних можна назвати SQLite.

Для визначення найкращих баз даних для створення телеграм-ботів необхідно порівняти їх за кількома ключовими критеріями, важливими для цього типу додатків. Потрібно дізнатись наскільки легко налаштувати та використовувати базу даних з кодом бота на різних мовах програмування. Визначити здатність бази даних ефективно обробляти зростаючі обсяги даних та кількість користувачів бота. Дослідити швидкість виконання операцій читання та запису, це важливо для забезпечення швидкої реакції бота. Окрім того є потреба у перевірці можливості легко змінювати структуру даних у міру розвитку бота. Порахувати витрати на ліцензування, хостинг та обслуговування бази даних. Також дуже важливим фактором є надійність та цілісність даних – гарантія збереження даних та їхньої коректності. Ну і наостанок, підтримка транзакцій важлива для забезпечення узгодженості даних при виконанні кількох пов'язаних операцій.

2.2 Обґрунтування вибору технологій

2.2.1 Найкращі бази даних для створення Telegram-бота

Вибір найкращої бази даних для телеграм-бота залежить від конкретних вимог проекту, його складності, обсягу даних та очікуваного навантаження. Тим не менш, можна виділити кілька оптимальних варіантів.

1. **SQLite –** база даних, яка використовується для невеликих та середніх ботів, які не потребують високої масштабованості або складної багатокористувацької взаємодії. Її головні переваги - простота використання, відсутність необхідності в окремому сервері та легка інтеграція з більшістю мов програмування. Вона зберігає дані у файлі, що робить її зручною для розгортання та перенесення. Підтримує ACID-транзакції, що гарантує цілісність даних на локальному рівні та значно покращує взаємодію користувача з системою.

2. **PostgreSQL –** це потужна та надійна реляційна база даних з відкритим вихідним кодом, яка є чудовим вибором для ботів середньої та великої складності, що потребують високої масштабованості, цілісності даних та підтримки складних SQL-запитів. PostgreSQL забезпечує повну підтримку ACID-транзакцій, має розвинені можливості для розширення та велику спільноту.

3. **MySQL –** популярна реляційна база даних, яка також підходить для ботів середньої та великої складності. Вона є широко використовуваною, має велику кількість інструментів та бібліотек для інтеграції. Продуктивність MySQL може бути високою при правильній конфігурації, а її масштабованість може бути досягнута за допомогою різних технік, таких як реплікація та кластеризація.

4. **MongoDB –** документо-орієнтована база даних NoSQL є гарним вибором для ботів, які працюють з неструктурованими або слабо структурованими даними, а також для проектів, які потребують високої гнучкості схеми та

горизонтального масштабування. Зберігання даних у вигляді документів спрощує розробку та дозволяє легко змінювати структуру даних у міру розвитку бота.

5. **Redis –** база даних типу "ключ-значення", яка відрізняється надзвичайно високою продуктивністю для операцій читання та запису. Redis часто використовується як кеш для зменшення навантаження на основну базу даних, для зберігання тимчасових даних (наприклад, станів користувачів у боті) або для реалізації простих черг повідомлень. Хоча Redis не підтримує складні SQL-запити або ACID-транзакції в повному обсязі, її швидкість робить її цінним інструментом для оптимізації продуктивності бота та значно полешує роботу у даній сфері.

2.2.2 Області використання різних баз даних

Для простих ботів з невеликим обсягом даних, які не потребують складної логіки або високої масштабованості, **SQLite** буде чудовим вибором завдяки своїй простоті та легкості інтеграції.

Для ботів середньої та великої складності, які потребують надійності, цілісності даних та підтримки складних запитів**, PostgreSQL** або **MySQL** є кращими варіантами. PostgreSQL часто вважається більш "просунутою" та гнучкою, тоді як MySQL має ширшу поширеність та більшу кількість доступних хостинг-провайдерів.

Якщо бот працює з великою кількістю неструктурованих даних або потребує високої гнучкості схеми та горизонтального масштабування**, MongoDB** може підійти найкраще.

Для підвищення продуктивності бота**,** кешування часто використовуваних даних або зберігання тимчасових даних, **Redis** є незамінним інструментом. Його можна використовувати в комбінації з іншою основною базою даних.

При виборі бази даних також слід враховувати знання та досвід команди розробників**,** доступність інструментів та бібліотек для обраної мови

програмування, а також бюджет проекту. Ретельний аналіз вимог проекту та порівняння різних варіантів допоможе зробити правильний вибір та забезпечити ефективну роботу телеграм-бота.

Було вирішено, що для найбільш продуктивного та зручного виконання моєї роботи найкраще підійде використання бази даних PostgreSQL.

2.2.3 Обґрунтування вибору PostgreSQL

Вибір PostgreSQL як бази даних є дуже обґрунтованим для ідеї створення інтелектуальної системи такого зразку, які були описані раніше. Були розглянуті головні причини, через які було обрано саме PostgreSQL і чому це буде найкращим рішенням для телеграм-бота, який буде вести облік спортивних досягнень співробітників:

Одною з головних причин вибору є підвищена надійність та цілісність даних. Для системи обліку досягнень, де важлива кожна зафіксована активність, надійність та цілісність даних є критично важливими. PostgreSQL повністю підтримує ACID-транзакції (Atomicity, Consistency, Isolation, Durability). Система гарантує, що кожна транзакція обробляється як єдина неподільна одиниця. Або всі зміни в межах транзакції фіксуються, або жодна з них не відбувається, що запобігає частковому оновленню даних. Наприклад, при реєстрації нового спортивного досягнення, всі пов'язані дані (працівник, вид спорту, час, дата) будуть записані разом, або не будуть записані взагалі.

Узгодженість забезпечує, що база даних завжди перебуває у валідному стані після будь-якої транзакції, дотримуючись усіх визначених правил та обмежень (наприклад, типи даних, унікальність ідентифікаторів).

Висока ізольованість гарантує, що одночасні транзакції не впливають одна на одну. Кожна транзакція виконується так, ніби вона є єдиною активною транзакцією в системі, що запобігає проблемам з одночасним доступом та оновленням даних. Це важливо, коли багато співробітників одночасно вносять свої дані. Після успішного завершення транзакції внесені зміни зберігаються

постійно і не будуть втрачені у разі збоїв системи (наприклад, вимкнення живлення).

Запропонована ідея передбачає необхідність фільтрації даних за різними критеріями: філіалами, регіонами, видами спорту та іншими параметрами. PostgreSQL надає потужний та гнучкий SQL, який дозволяє виконувати складні аналітичні запити.

В ході роботи, буде можливість легко отримувати статистику по кожному виду спорту в конкретному філіалі за певний період часу. Окрім того, завдяки універсальності та гнучкості PostgreSQL можна буде будувати рейтинги співробітників на основі їхніх досягнень, фільтруючи за різноманітними критеріями.

SQL дозволяє об'єднувати дані з різних таблиць (наприклад, таблиці співробітників та таблиці досягнень) для отримання комплексної інформації. Присутня підтримка різноманітних агрегатних функцій (SUM, AVG, COUNT, MAX, MIN), вона дозволить нам легко обчислювати підсумкові показники.

PostgreSQL підтримує широкий спектр вбудованих типів даних, а також дозволяє створювати власні типи. Це може бути корисним для зберігання специфічної інформації про спортивні досягнення.

Якщо виникне потреба зберігати напівструктуровані дані (наприклад, додаткову інформацію про змагання або індивідуальні результати), PostgreSQL чудово справляється з цим завдяки підтримці JSON та його бінарного формату JSONB, який забезпечує ефективне індексування та запити.

Цікаво, що система містить географічні розширення. Хоча безпосередньо для обліку спортивних досягнень це може не знадобитися на початковому етапі, у майбутньому, якщо буде потреба, наприклад, організовувати спільні спортивні заходи на певних локаціях, розширення PostGIS надасть потужні інструменти для роботи з геопросторовими даними.

Мабуть головною причиною вибору PostgreSQL є масштабованість. Адже на початковому етапі кількість користувачів може бути невеликою, а от з ростом компанії та популярності моєї інтелектуальної системи, кількість

записів та обсяг даних будуть зростати. PostgreSQL є високомасштабованою базою даних, яка може ефективно обробляти великі обсяги інформації та високе навантаження.

Підтримка індексації дозволяє оптимізувати швидкість виконання запитів навіть при великій кількості даних. Можливості реплікації та кластеризації забезпечують високу доступність та розподіл навантаження у випадку дуже великих компаній з численними філіалами.

PostgreSQL є однією з найпопулярніших баз даних з відкритим вихідним кодом. Це означає наявність великої та активної спільноти розробників, яка постійно працює над її вдосконаленням, виправленням помилок та створенням різноманітних інструментів та розширень.

Існує велика кількість документації, навчальних матеріалів та готових бібліотек для різних мов програмування, що значно полегшить процес розробки та інтеграції телеграм-бота з PostgreSQL.

PostgreSQL є безкоштовною у використанні, оскільки розповсюджується під ліцензією PostgreSQL License, яка є дуже ліберальною. Це дозволить заощадити кошти на ліцензіях.

Враховуючи необхідність надійності, цілісності даних, можливості складних фільтрацій та аналізу, а також масштабованість задуманого проєкту, PostgreSQL є найкращим вибором бази даних для телеграм-бота обліку спортивних досягнень співробітників. Її потужні можливості SQL, підтримка ACID-транзакцій, розширені типи даних та зріла екосистема забезпечать стабільну, ефективну та гнучку основу для вашої інформаційної системи.

2.3 Переваги та недоліки обраних технологій

Створення нової інтелектуальної системи зі своїми нормами та своїми правилами – непростий процес, який потребує серйозного підходу. Були довгі вагання, щодо того, які технології треба використовувати в роботі, проте в результаті вибір пав на Telegram-бот, як основу для роботи та в якості бази

даних, яка буде наповнюватись, вибір зупинився на реляційній базі даних PostgreSQL.

2.3.1 Переваги та недоліки Telegram-боту як інтелектуальної системи

У контексті створення сучасної інформаційної системи для обліку спортивних досягнень працівників, використання Telegram-бота як інтелектуальної системи є надзвичайно вдалим вибором. Telegram – одна з найпопулярніших платформ обміну повідомленнями у світі, яка активно використовується мільйонами людей, зокрема й у корпоративному середовищі. Це робить бот у Telegram зручним і доступним інструментом, який не потребує встановлення додаткового програмного забезпечення.

Серед основних переваг такого бота як рішення для взаємодії з користувачами варто виділити доступність і простоту використання. Практично кожен співробітник має смартфон із встановленим месенджером Telegram. Це означає, що не потрібно створювати окремий застосунок або сайт, адже бот доступний у декілька кліків. Telegram працює на всіх основних операційних системах — Android, iOS, Windows, macOS та навіть має вебверсію. Таким чином, бот буде доступний будь-де, де є інтернет. Бот може бути інтегрований із зовнішніми системами, такими як бази даних, сервіси статистики, аналітики, системи аутентифікації тощо. Окрім того, за допомогою бота можна автоматично приймати дані від користувачів, обробляти їх, виводити звіти та статистику, відправляти повідомлення, нагадування тощо. Розробка бота в Telegram значно дешевша, ніж створення повноцінного застосунку чи вебпорталу, особливо на початкових етапах. Telegram має високий рівень безпеки, включаючи контроль доступу, шифрування та автентифікацію, що важливо при роботі з персональними даними.

Проте, попри перелічені переваги, існують і певні недоліки, які слід враховувати. Бот працює в рамках текстового чату, тому складна візуалізація

даних або інтерактивні елементи (графіки, діаграми тощо) мають суттєві обмеження. У порівнянні з повноцінними застосунками, Telegram-бот має обмеження в складних сценаріях навігації, що може ускладнити реалізацію деяких функцій. До того ж, Telegram – сторонній сервіс, який у будь-який момент може змінити політику доступу або API, що може вплинути на роботу бота. Для забезпечення стабільної роботи необхідно постійно слідкувати за оновленнями Telegram API та змінювати код у разі потреби.

Але переваг більше ніж недоліків! Таким чином, Telegram-бот є зручною, сучасною та економічно ефективною платформою для реалізації інтелектуальної системи, але для розширення функціональності або у випадку складної логіки взаємодії він може потребувати додаткових компонентів.

2.3.2 Переваги та недоліки PostgreSQL як основної бази даних

Однією з ключових складових інформаційної системи для обліку спортивних досягнень працівників є надійна система зберігання, обробки та фільтрації даних. У цьому контексті вибір бази даних PostgreSQL є цілком виправданим і обґрунтованим.

PostgreSQL — це потужна об'єктно-реляційна система управління базами даних з відкритим вихідним кодом, яка широко використовується у різних сферах: від малих стартапів до великих корпорацій. Її можливості дозволяють ефективно реалізовувати складні структури даних, запити та логіку, необхідну для функціонування бота.

PostgreSQL найкраще підходить для реалізації моєї ідеї. Серед основних переваг можна виділити підтримку складних запитів, масштабованість, надійність та інші важливі причини. PostgreSQL дозволяє працювати з великими обсягами даних, підтримуючи складні SQL-запити, фільтрацію, агрегацію, сортування та інші операції, що є надзвичайно корисним при побудові статистики за спортивними досягненнями. База підтримує створення користувацьких типів даних, функцій, розширень, що дозволяє адаптувати

систему під будь-які специфічні потреби компанії. PostgreSQL має вбудовану підтримку транзакцій, контроль цілісності, зовнішні ключі, обмеження та інші механізми, що забезпечують надійність і стабільність роботи системи. Система добре справляється з великими обсягами даних, що робить її придатною як для невеликих компаній, так і для корпоративного використання. PostgreSQL має багаторівневу систему контролю доступу, шифрування підключень, а також можливість гнучко управляти правами користувачів.

Для реалізації телеграм-бота зазвичай використовують Python. PostgreSQL чудово інтегрується з ним через бібліотеки (наприклад, psycopg2), що спрощує роботу з базою.

Разом з тим, існують і недоліки, які потрібно враховувати. В якості приклада можна навести складність налаштування та адміністрування. PostgreSQL потребує певних знань для встановлення, налаштування та оптимізації, особливо якщо планується масштабне використання або розподілене середовище. У порівнянні з деякими легшими СУБД, PostgreSQL може споживати більше ресурсів, особливо при роботі з великими даними та складними транзакціями.

Підсумувавши всі переваги та недоліки, можна впевнено сказати, що переваг набагато більше і вони суттєвіші за недоліки. З урахуванням цього, потрібно зауважитти, що PostgreSQL є надійним, продуктивним та безпечним рішенням для зберігання та обробки даних у рамках розробки інтелектуальної системи Telegram-бота для моніторингу спортивної активності працівників компанії.

2.3.3 PostgreSQL порівняно з іншими базами даних

SQLite:

SQLite є дуже простою у використанні та чудово підходить для невеликих локальних додатків, проте вона не призначена для високого навантаження та

одночасного доступу великої кількості користувачів, що може стати проблемою для великої компанії з багатьма співробітниками, які активно користуються ботом.

MySQL:

MySQL також є гарним варіантом реляційної бази даних, але історично PostgreSQL вважається більш суворою у дотриманні стандартів SQL та має деякі розширені можливості, які цілком можуть бути корисними для даного проекту.

MongoDB (NoSQL):

Хоча NoSQL бази даних, такі як MongoDB, пропонують гнучкість схеми, для задачі, де структура даних є досить чіткою (працівник, вид спорту, час, дата тощо) і потрібні складні фільтрації та агрегації, реляційна модель PostgreSQL з її потужним SQL може бути більш ефективною та зручною.

Redis (Key-Value):

Redis чудово підходить для кешування та швидкого доступу до простих даних, але вона не призначена для зберігання структурованих даних з складними зв'язками та потребою у складних запитах, які ви передбачаєте у своєму боті.

Висновки до розділу 2

Було здійснено порівняльний аналіз сучасних підходів, технологій та інструментів, що застосовуються для створення інтелектуальних Telegram-ботів, зокрема таких, які використовуються для обліку й стимулювання спортивної активності співробітників компаній.

Проведене дослідження показало, що Telegram-боти є зручною, масштабованою та ефективною формою взаємодії користувачів з інформаційною системою. Вони не потребують встановлення додаткового програмного забезпечення, працюють у вже знайомому користувачеві середовищі та легко інтегруються з зовнішніми сервісами й базами даних.

Окрему увагу було приділено способам інтеграції Telegram-бота з базами даних, оскільки саме завдяки цьому реалізується можливість зберігати, аналізувати та виводити інформацію про спортивну активність користувачів. Було розглянуто пряме підключення до бази даних, створення окремого API-сервісу для взаємодії та використання конструкторів ботів із вбудованою підтримкою баз. Найбільш перспективним і гнучким підходом, особливо з погляду безпеки та масштабованості, є розробка окремого BackEnd API, який забезпечує контрольований і стандартизований доступ до бази даних.

У порівнянні між типами баз даних було встановлено, що реляційні СУБД, зокрема PostgreSQL, є найкращим вибором для даного проєкту. Це зумовлено високою надійністю, підтримкою складних запитів, транзакційністю, масштабованістю та широкими можливостями для розширення (наприклад, підтримка JSON, робота з просторовими даними тощо).

Таким чином, на основі аналізу існуючих рішень, технологій і підходів, можна зробити висновок, що найоптимальнішою архітектурою для реалізації інтелектуальної системи моніторингу спортивної активності працівників є Telegram-бот, побудований на базі спеціалізованого фреймворку з інтеграцією до реляційної бази даних PostgreSQL через окремий BackEnd API. Така

структура забезпечить гнучкість, масштабованість, зручність у користуванні та високий рівень автоматизації, що відповідає як поточним технічним вимогам, так і потенційному майбутньому розширенню системи.

У процесі розробки інтелектуальної системи для автоматизованого обліку спортивної активності працівників компанії було проаналізовано низку технологій, платформ і рішень. В результаті обґрунтованого технічного та функціонального аналізу було обрано Telegram-бот як основний інтерфейс взаємодії користувача із системою, а також реляційну базу даних PostgreSQL для зберігання, обробки та фільтрації інформації.

Telegram-бот виявився оптимальним рішенням з огляду на його доступність, кросплатформеність, високу популярність серед користувачів, простоту використання, невисоку вартість розробки та легкість у розгортанні. Він дозволяє реалізувати зручний і зрозумілий інтерфейс для взаємодії з користувачами без необхідності створення окремого мобільного додатку чи вебпорталу. Крім того, Telegram-бот надає можливість інтеграції з іншими сервісами та обробки даних у режимі реального часу, що робить його ефективним інструментом для збору інформації про спортивні досягнення працівників.

Водночас для забезпечення надійного зберігання даних було обрано PostgreSQL — потужну, масштабовану та безпечну систему управління базами даних. PostgreSQL підтримує складні SQL-запити, транзакції, зовнішні ключі, цілісність даних, а також має гнучкі механізми управління доступом. Її гнучкість та розширюваність дозволяють ефективно адаптувати систему до потреб конкретної компанії, включаючи можливість побудови звітів за філіалами, регіонами, видами спорту та іншими параметрами. У порівнянні з альтернативами, такими як SQLite, MySQL, MongoDB або Redis, PostgreSQL забезпечує найкращий баланс між функціональністю, продуктивністю та стабільністю, що є критично важливим для корпоративного рівня застосування.

Таким чином, комбінація Telegram-бота як інтерфейсу користувача та PostgreSQL як основи для обробки й зберігання даних дозволяє створити зручну, ефективну та сучасну інформаційну систему. Такий підхід повністю відповідає цілям і задачам дипломного проєкту, а також задовольняє вимоги компаній, які прагнуть заохочувати здоровий спосіб життя своїх співробітників шляхом системного моніторингу їхньої спортивної активності. Це рішення є не лише технічно обґрунтованим, але й практично орієнтованим, з чіткою перспективою впровадження у реальних умовах.

РОЗДІЛ 3

ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ

3.1 Проектування архітектури системи

Після визначення підходу до розробки інтелктуальної системи, наступною задачею є проектування архітектури системи.

Дана інтелектуальна система повинна складатися з серверної та клієнтської частини. В якості технології створення серверної частини інтелектуальної системи було обрано варінт побудови за допомогою бази даних. Клієнтська частина, в свою чергу, розділяється ще на дві: користувацьку та адміністративну.

3.1.1 Користувацька частина

Основним призначенням даної інтелектуальної системи є можливість безперешкодно користуватися нею звичайним клієнтам. Перед будь-яким користувачем, який не володіє правами адміністратора, буде відкриватися можливість використовувати інтелектуальну систему лише в користувацькій частині.

Оскільки система призначена для фіксації спортивних результатів працівниками компанії, то і користуватись нею будуть здебільшого співробітники різноманітних фірм та компаній. Працівники які будуть залучені до занять спортом, після кожного свого тренування з бігу та поїздкою на велосипеді будуть мати можливість записати свої резльтати, такі як час та дату початку тренування, дисципліну, тривалість та подолану дистанцію в ході тренування.

На даний момент інтелектуальна система створена лише для таких дисциплін, як біг, їзда на велосипеді та дуатлон – вид спорту, який передбачає трьохетапне чергування двох спортивних дисциплін: бігу та їзди на велосипеді

(біг + велосипед + біг). Проте це лише пробна версія даного продукту і в майбутньому планується розширення з додаванням великої кількості інших спортивних дисциплін.

Важливо зауважити, що пепередній сценарій актуальний виключно для зареєстрованих користувачів. Проте, якщо людина заходить в систему вперше, то вона повинна пройти етап реєстрації перед тим, як отримати можливість фіксувати свої результати з спортивних дисциплін.

В ході реєстрації кожен користувач повинен вказати своє повне ім’я, прізвище та по батькові – це потрібно адміністраторам для фільтрації користувачів по імені, а також для фіксації результатів користувача в базі даних саме під його іменем. Окрім того, варто вказати в якому регіоні працює користувач, в якій компанії та за якою спеціальністю. Ці дані знадобляться для фільтрації інформації по регіонам. Також буде потрібно вказати номер телефону, електронну пошту, спортивну категорію, спорт, яким займається користувач та спорт, яким планує займатись.

Після проходження процедури деєстрації, кожен користувач може переходити до заповнення своїх спортивних результатів.

3.1.2 Адміністративна частина

Інформацію про спортивні досягнення, яку заповнюють користувачі, хтось повинен систематизувати та фільтрувати за різними характеристиками. Для цього в даній інтелектуальній системі обов’язково повинен бути адміністратор.

В якості адміністратора, компанія має призначити одного або кількох відповідальних співробітників компанії, які будуть слікувати за спортивною активністю працівників. Ідентифікаційний номер акаунту цього співробітника має бути вказаний в системі і в такому разі при запуску вмикається саме адміністративна частина.

Адміністратор може створювати різноманітні запити до бази даних, з метою формування узагальнених відомостей щодо спортивних досягнень співробітників компанії по певних критеріям.

Можна вивести повну базу учасників, незалежно від філіалу, регіону та прізвища. Проте, якщо дана система буде використовуватися на рівні цілої країни та у всеукраїнських компаніях, то повна база учасників буде занадто масштабною, для цього потрібно виводити базу учасників по різним критеріям.

Враховуючи, що великі компанії, як правило, розташовані в різних регіонах, то було прийнято рішення, одним із можливих запитів, зробити запит по регіонам. Серед можливих регіонів – Київщина, Полтавщина, Сумщина, Чернігівщина, Житомирщина, Харківщина, Тернопільщина, Хмельниччина, Волинь, Вінниччина, Черкащина, Рівненщина та Львівщина.

Також є можливість виводити інформацію по прізвищам. Це допоможе дізнатися про спортивну активність конкретного співробітника за весь час. Окрім того є можливість фільтрувати по даті. Можна вивести інформацію про спортивні досягнення всіх працівників за якусь конкретну дату та відфільтрувати інформацію за певний діапазон часу (наприклад, від 28.05.2025 до 30.05.2025).

А ще можна робити фільтрацію за видом спорту (велоспорт, біг або дуатлон) та за компанією.

Окрім всіх вищеперерахованих категорій, можна просто вивести базу всіх учасників – користувачів, які зареєструвались в інтелектуальній системі.

3.2 Вибір алгоритмів та методів обробки даних

В ході вибору алгоритмів та методів обробки даних, було проведено детальний та дуже прискіпливий аналіз всіх «за» і «проти» за ту чи іншу технологію.

3.2.1 Вибір бази даних

В даній інтелектуальній системі в якості бази даних використовуємо PostgreSQL. Користувачі мають можливість тільки вносити інформацію в базу даних. В той час як адміністратори можуть не тільки вносити, а ще й отримувати в зручному вигляді та аналізувати отриману інформацію.

PostgreSQL є оптимальним вибором для створення інтелектуальної системи для фіксації спортивних досягнень працівників компанії. Ця база даних гарантує цілісність та збереження кожного заняття спортом завдяки підтримці ACID-властивостей. Тобто внесені дані завжди будуть повними і точними.

PostgreSQL підтримує широкий спектр типів даних і це дозволяє моделювати будь-які спортивні показники. А завдяки системі роширень та можливості створювати власні функції, можна легко інтегрувати складну логіку та аналітику, що робить новостворену систему по-справжньому інтелектуальною!

Також користувачам пропонується розширений SQL з віконними функціями та оптимізатором запитів. Це дозволяє проводити глибокий аналіз даних, виявляти тенденції, будувати рейтинги та формувати звіти про прогрес працівників швидко і дуже ефективно.

Система легко масштабується для обробки великих обсягів даних та великої кількості користувачів, що є дуже важливим для зростаючої компанії.

Також важливо зауважити, що PostgreSQL є безкоштовним, а його активна спільнота забезпечує постійну підтримку та розвиток, знижуючи вартість володіння та знижуючи ризики.

Ці переваги роблять базу даних PostgreSQL напрочуд міцною та дуже гнучкою основою для інтелектуальної системи, яка не лише фіксує досягнення, а й сприяє мотивації для занять спортом для кожного працівника компанії.

3.2.2 Вибір мови програмування

Для створення програмного коду інтелектуальної системи було обрано мову програмування Python.

Дана мова має легкий та зрозумілий синтаксис, що дозволяє писати чистий і читабельний код. Це значно прискорює процес розробки та спрощує підтримку проєкту. Менше коду – менше помилок! Це створює швидке впровадження нових функцій, що критично важливо для інтерактивної системи.

Python має одну з найбільших і найактивніших екосистем бібліотек та фреймворків, які є головими рішеннями для різних завдань. Деякі фреймворки дозволяють швидко створити надійний і масштабований бекенд для створеної системи, забезпечуючи взаємодію з базою даних та користувацьким інтерфейсом.

Для створення інтелектуальної частини системи Python не має собі рівних. Бібліотеки надають потужні інструменти для обробки та аналізу спортивних показників Також існують бібліотеки, які дозволяють створювати красиві та інформативні графіки та діаграми спортивного прогресу, що є ключовим для мотивації працівників. Окрім того є бібліотеки для прогнозування результатів, виявлення тенденцій у спортивних данних або навіть персоналізованих рекомендацій щодо тренувань.

Python є кросплатформною мовою, тобто код, написаний на цій мові програмування, може працювати на різних операційних системах без змін. Це дає гнучкість у розгортанні та підтримці системи.

В ході написання програмного коду часто виникає багато труднощів. Величезна спільнота користувачів Python допомагає завжди швидко знаходити відповіді на всі питання і вирішувати будь-які проблеми, отримувати готові рішення для типових проблем та безліч навчальних матеріалів. Це робить процес розробки більш ефективним та приємним.

Python легко інтегрується з іншими технологіями, зокрема з різними API та базами даних, як-от PostgreSQL, про яку було сказано раніше. Це дозволяє легко підключати зовнішні джерела даних або розширювати функціонал системи.

Завдяки поєднанню простоти, потужної екосистеми та широких можливостей для аналізу даних і машинного навчання, Python є ідеальним вибором для сучасної інтелектуальної системи фіксації спортивних досягнень співробітників компанії

3.2.3 Вибір програми для програмування

Весь програмний код реалізації програми було створено в застосунку PyCharm Professional. Цей застосунок є одним з найкращих інтегрованих середовищ розробки (IDE), яке значно підвищить продуктивність та якість коду.

PyCharm Professional є чудовим вибором для проєкту, зокрема, через інтелектуальну допомогу в кодуванні. Штучний інтелект, який вбудований у програму, самотужки передбачає подальші дії автора коду, і надає контекстно-залежні підказки для змінних, функцій, методів, класів та модулів. Це неабияк економить час, зменшує кількість помилок та прискорює процес написання коду. IDE постійно перевіряє написаний код на наявність синтаксичних помилок, потенційних проблем та порушень стандартів кодування. Система підсвічує їх і пропонує швидке виправлення. PyCharm Professional надає потужні можливості для автоматичного рефакторинга коду (перейменування змінних/функцій, винесення фрагментів коду в окремі методи, зміни сигнатур функцій тощо). Все це дозволяє легко покарщувати структуру коду без ризику внесення нових помилок.

Саме версія Professional дозволяє гарно інтегруватись з базами даних! Для системи фіксації досягнень потрібна взаємодія з базою даних. У випадку описуваної інтелектуальної системи, була задіяна база даних PostgreSQL.

PyCharm Professional має вбудовані інструменти для роботи з базами даних, які дозволяють підключатися до баз даних, переглядати системи, таблиці та дані. Також можна писати та використовувати SQL-запити безпосередньо з IDE. Існує автодоповнення SQL-коду та підсвічування синтаксису для SQL. Дуже якісно створені інструменти для міграції схем та генерації коду для роботи з базою даних (ORM). Це значно спрощує розробку програмного коду та взаємодію з даними.

Одною з головних “фішок” PyCharm Professional є інструменти для наукових обчислень та машинного навчання. Саме тут застосунок розкриває свій інтелектуальний потенціал, що є критично важливим для проєкту. Доволі якісно реалізовано інтеграцію з Jupyter Notebook. Можна створювати, редагувати та запускати Jupyter Notebook прямо в PyCharm. Це ідеально підходить для інтерактивного аналізу даних, експериментів з алгоритмами машинного навчання та візуалізації результатів. В PyCharm Professional розширена підтримка наукових бібліотек. Це означає краще автодоповнення, інспекції та можливість переглядати дані у спеціальних вікнах. Вбудовані інструменти для візуалізації графіків та діаграм, створених за допомогою Matplotlib або Plotly, що допомагає краще розуміти спортивні дані та прогрес користувачів.

Враховуючи, що в якості мови програмування було обрано Python, то PyCharm Professional було обрано недарма. PyCharm має один з найкращих візуальних відладчиків для Python. Він дозволяє встановлювати точки зупинки, покроково виконувати код, переглядати значення змінних в реальному часі, аналізувати стек викликів та відлагоджувати веб-додатки та навіть код у Jupyter Notebook. Це значно прискорює пошук та виправлення помилок, що є надзвичайно важливим для складних систем.

PyCharm Professional об’єднує багато інструментів в одному місці. Має глибоку інтеграцію з Git та іншими VCS для легкого управління версіями коду, створення комітів та злиття гілок. Існує вбудований термінал для виконання команд, роботи з віртуальними середовищами та встановлення пакетів. Також

в PyCharm Professional є інструменти для аналізу продуктивності коду, які допоможуть оптимізувати роботу системи

Функціональність та значна економія часу під час розробки та налагодження роблять його вигідною інвестицією для професійного проєкту, який вимагає інтелектуальних функцій, аналізу даних та надійної взаємодії з базою даних. Для освітніх цілей або невеликих проєктів може бути достатньо PyCharm Community, але для створення повноцінної інтелектуальної системи з інтеграцією бази даних PyCharm Professional надає набагато більше можливостей.

3.2.4 Формування відповіді

Кожен офісний працівник хоча б раз стикався з застосункою Microsoft Excel, тому не буде проблем з тим, аби навчити людину, яка буде відповідальним адміністратором. Це означає, що їм не знадобиться додаткове навчання для перегляду звітів про спортивні досягнення, що робить інформацію легкодоступною та зрозумілою для широкого кола користувачів.

Excel чудово підходить для створення чітких і структурованих таблиць, які можуть відображати рейтинги, особисті рекорди, статистику тренувань та інші дані. Користувачі можуть легко сортувати, фільтрувати або навіть виконувати базові розрахунки безпосередньо у файлі.

Файли, створені в Excel легко поширювати результати спортивних досягнень від адміністраторів своїм керівникам.

Python-система з PostgreSQL може генерувати та експортувати дані в Excel-файли. Це дозволить користувачам завантажувати ці файли для перегляду онлайн, друку або подальшого самостійного аналізу.

Комбінація всіх інструментів інтелектуальної системи, таких як мова програмування Python, база даних PostgreSQL та виведення результатів в таблиці Excel, забезпечує потужну систему з дружнім для користувача способом представлення інформації.

3.3 Розробка інтерфейсу користувача

В якості інтерфейсу пропонується Телеграм-бот для користувача і адміністратора для доступа до бази даних.

При вході в бот інтелектуальна система визначає статус клієнта (адміністратор або користувач) по його ідентифікайному номеру в застосунку Telegram. Для різного статуса користувача використовуються різні форми. Вище, в пункті 3.1 було детально розписано чим відрізняється бот для адміністратора та бот для користувача.

3.3.1 Результати помилкового введення в формі реєстрації

Для користувача використовується спеціальна форма запитів для збору інформації для реєстрації або фіксації результатів спортивних досягнень працівників компанії. Система має змогу аналізувати введену інформацію користувачем і в разі некоректного вводу інформації виправляє користувача і просить ввести правильні дані.

В формі реєстрації для користувача, людина, яка заповнює особисту інформацію не має можливість ввести будь-що в пунктах «прізвище», «ім’я», «по батькові», «компанія», «спеціальність», «номер телефону», «адреса електронної пошти» тощо.

Проте, є такі пункти, які не можна вводити. Там дається змога обрати відповідь серед наведених варіантів. Наприклад, коли користувач обирає спорт, яким займається, то він отримує три варіанта можливої відповіді (БІГ, ВЕЛО, ДУАТЛОН). Якщо користувач напише щось своє, що не буде збігатись з одним із наданих варіантів, то у відповідь отримає повідомлення від бота з наступним текстом: *«Виберіть з наданих варіантів».*

Також аналогічна ситуація складається, коли користувач обирає регіон. Він також може обрати один із варіантів, в даному випадку він отримує 13 варіантів для вибору регіона в яких функціонує його компанія (Київщина,

Полтавщина, Сумщина, Чернігівщина, Волинь, Житомирщина, Харківщина, Тернопільщина, Хмельниччина, Вінниччина, Черкащина, Рівненщина, Львівщина). Якщо користувачем буде введено якийсь інший регіон або будь-який інший варіант, окрім запропонованих, то у відповідь буде запропоновано знову обрати серед попередньо наданих варіантів та отримано повідомлення від бота *«Виберіть регіон серед наданих варіантів».*

3.3.2 Результати помилкового введення в формі досягнень

Коли співробітник компанії зареєструвався та вказав необхідні дані. Перед ним відкривається можливість заповнювати інформацію щодо проведених тренувань та спортивної активності.

Кожному зареєстрованому користувачу для того, що зафіксувати результати необхідно спершу обрати вид спорту, з якого проходило заняття, серед можливих варіантів (БІГ, ВЕЛО, ДУАТЛОН). Якщо ввести щось, що не буде відповідати одному з наведених варіантів, то користувач отримає можливість знову обирати серед попередньо наданих варіантів та відповідь від бота *«Виберіть серед наданих варіантів».*

Після того як спорт буде обраний, то користувач повинен буде ввести дату заняття (зазвичай це поточна дата) в форматі DD.MM.25 (де DD – це десь місяця, а MM – місяць по номеру), наприклад, 29.05.25. Якщо дата буде введена в неправильному форматі або буде введено неіснуючу дату (наприклад, 37.15.25), то користувач отримає у відповідь від бота повідомлення *«Виберіть дату в форматі DD.MM.25».*

Після введення дати в правильному форматі, необхідно ввести час початку заняття в форматі ГГ:ХХ (де ГГ – години, а ХХ – хвилини. Аналогічно, як і при введенні дати, якщо користувач буде вводити час в неправильному форматі (наприклад, 22-15) або буде зазначено некоректний та неіснуючий час (наприклад, 27:82), то система не пропустить цю відповідь,

попросить ввести знову та у відповідь на неправильний ввід, напише повідомлення *«Введіть час в форматі ГГ:ХХ»*.

Коли на попередні питання надана корестна відповідь, то користувач потрібен ввести результат подоланої дистанції під час бігу або дистанцію, яку було подолано на велосипеді, в метрах, виключно цифрами. Якщо буде введено символи, окрім цифр, наприклад, букви або розділові знаки, тоді система не запише результат, попросить його ввести знову за допомогою повідомлення *«Введіть дистанції в метрах виключно цифрами»*.

Після цього користувачем вказується час подоланої дистанції в форматі ГГ:ХХ:СС (де ГГ – години, ХХ – хвилини, СС – секунди), незалежно від виду спорту, яким займався учасник. В разі неправильного формату вводу користувач отримає повідомлення *«Введіть час в форматі ХХ:ХХ:ХХ»*.

Наостанок, після того як вид спорту, дата тренування, час початку тренування, подолана дистанція були введені, то залишається останнє – треба підкріпити свій результат за допомогою світлини. Оскільки кожен користувач може вводити будь-які дані, то існує ймовірність, що працівник міг ввести вигадану інформацію, яка не відповідає дійсності. Для цього було впроваджено систему, яка зменшує можливості обманювати адміністраторів фейковими даними. Останнім кроком перед записом результатів в базу даних є фотографія. На цій фотографії має бути підтвердження введених даних. Це може бути фото з фітнес-трекера, скріншот з програми для занять спортом тощо. Якщо фото не було надіслано, то користувач отримає повідомлення *«Ви не надіслали фото»*.

3.3.3 Структура. Тека Handlers

Для загальної програми було розроблено певну структуру в якій файли були розподілені по текам залежно від їхнього функціонального призначення.

Структуру було вирішено побудувати в вигляді кількох каталогів з різними файлами.

Фактично тека Handlers відповідальна за форми запитів. Програмний код цієї теки описує основні функціональні можливості Telegram-бота, призначеного для керування реєстрацією користувачів, відстеження спортивних результатів та надання адміністративних можливостей запитів.

Бот чітко розділяє зони відповідальності, де файл **user_commands.py** керує початковою взаємодією користувачів та командами. Зокрема саме в цьому файлі відбувається обровка команди */start*. Коли користувач запускає бот, він спочатку налаштовує команди бота за допомогою *set_commands*. Потім він надсилає сповіщення до *ID_ADMIN* та *ID_SUPERADMIN* про запуск бота певним користувачем. Саме тут основна логіка визначає статус користувача. Якщо Telegram ID користувача збігається з *ID_ADMIN* або *ID_SUPERADMIN*, то його вітають як адміністратора та пропонують опцію «АДМІНІСТРУВАННЯ». У випадку, якщо користувач не є адміністратором, то система перевіряє чи існує його ідентифікатор у базі даних за допомогою *get_base_telegramID*. В разі, якщо користувача не знайдено, то йому пропонується зареєструватися за допомогою кнопки *«РЕЄСТРАЦІЯ»*. Ну а якщо користувач вже зареєстрований, тоді його вітають та пропонують кнопку «ФІКСАЦІЯ РЕЗУЛЬТАТІВ»

В каталозі Handlers є файл **form_admin_query.py**, який відповідає за обробку адміністративних запитів в боті. Коли користувач надсилає команду «адміністрування», тоді бот переходить в стан, де пропонує адміністратору вибрати критерій запиту. Серед наданих варіантів є «База учасників», «Прізвище», «Регіон», «Компанія», «Вид спорту» та «Дата». Залежно від обраного критерію, бот проводить адміністратора через подальші кроки для уточнення запиту. В свою чергу, запити виконуються за допомогою функцій з файла *db_utils* з теки *utils* (цю теку буде розглянуто дельніше згодом) для отримання даних з бази та надсилання результатів адміністратору. Також передбачена обробка помилок для некоректних введень, таких як неправильні формати дати або вибір поза наданими опціями.

Окрім попередніх файлів, саме до Handlers відносяться файли для користувачів. Зокрема, файл **form_registration.py** керує процесом реєстрації користувачів для бота. Коли користувач надсилає команду «реєстрація», бот приводить його через низку запитань для збору особистої та професійної інформації. Це включає прізвище, ім'я, по батькові, регіон (вибраний зі списку), компанію, спеціальність, номер телефону, електронну адресу та деталі щодо спортивної діяльності. Після зрору всієї необхідної інформації вона зберігається в базі даних за допомогою *add_base_empl*. Підсумок записаних даних нового зареєстрованого користувача надсилається одразу до *ID_ADMIN* та *ID_SUPERADMIN* для адміністративного контролю. Файл також включає, описану в розділі 3.1, перевірку вибору регіону та виду спорту.

Останній файл, який міститься в теці під назвою Handlers – це файл **form_athletics.py**. Цей обробник зосереджений на фіксації результатів спортивних досягнень користувачів. Після отримання команди «фіксація результатів» бот ініціює керовану розмову для збору деталей про тренування користувача. Він послідовно запитує вид спорту, дату, час початку, дистанцію та кінцевий результат. Важливо, що саме він також пропонує завантажити користувачеві фотографію свого тренування. Після збору та перевірки всієї інформації, дані, включно з фото, зберігаються в базі даних за допомогою функцій *add_base_athl* та *get_photo*. Підсумок записаних результатів надсилається на заздалегідь визначені індентифікатори для адміністративного контролю для сповіщення. Файл також містить обробку помилок для некоректних форматів введення на кожному етапі.

Фактично інтелектуальна система дійсно чітко розділяє зони відповідальності, таких як реєстрація користувачів, відстеження спортивних досягнень та адміністрування

Ключові утиліти використовуються для взаємодії з базою даних, для динамічного створення вбудованих клавіатур та визначення різних сианів у рамках FSM. Споввіщення адміністраторам є постійною функцією під час реєстарції та фіксаіції результатів, забезпечуючи контроль. А створений файл

user_commands.py діє як точка входу, направляючи користувачів на основі їхнього адміністративного статусу реєстрації, забезпечуючи індивідуальний початковий досвід.

3.3.4 Структура. Тека Utils

Тека Utils містить основні допоміжні компоненти, які підтримують функціональність бота. Вона охоплює такі важливі аспекти, як керування базою даних, обробка даних, реєстрація команд та керування станами для потоків розмов.

Файл **commands.py** забезпечує базову реєстрацію команд, роблячи бота зручним для користувача, перелічуючи доступні команди. Цей файл досить простий і зосереджений на визначенні та налаштуванні стандартних команд для Телеграм-бота. Він містить асинхронну функцію *set_commands*, яка приймає екземпляр *Bot* як аргумент. У цій функції створюється список об'єктів *BotCommand*. Наразі він включає лише команду */start* з описом «Початок роботи». Цей список команд потім надсилається до Telegram API, роблячи ці команди видимими та доступними в інтерфейсі чату бота. Це допомагає користувачам легко знаходити основні функції.

Файл **db.py** налаштовує реляційну структуру бази даних за допомогою SQLAlchemy, визначаючи, які дані співробітників та спортивні результати зберігаються та пов'язуються. Цей файл є визначенням схеми бази даних, він ініціалізує *Base* для декларативних моделей. *BaseModel* слугує абстрактним базовим класом для інших моделей, надаючи спільний первинний ключ *id*.

MyEmployees визначає схему для зберігання даних реєстрації користувачів. Вона включає стовпці для особистих даних, унікальний telegramID та інформацію, пов'язану зі спортом (основний вид спорту, бажаний вид спорту, категорія). Вона також встановлює зв'язок "один до багатьох" з записами MyAthletics, вказуючи, що один співробітник може мати кілька спортивних результатів.

MyAthletics визначає схему для зберігання результатів спортивних тренувань. Вона включає стовпці для дисципліни, дати, часу, дистанції та результату. Вона має зв'язок "багато до одного" з *MyEmployees* через *employee_id*, пов'язуючи кожен спортивний запис з певним співробітником компанії. Файл налаштовує з'єднання з базою даних PostgreSQL за допомогою *create_engine* і створює всі визначені таблиці в базі даних, якщо вони ще не існують.

Файл **db_utils.py** діє як основниний інтелфейс для всіх операцій з базою даних, пропонуючи функції для додавання нових записів, отримання конкретної інформації про користувача та, що найважливіше, генерації динамічних звітів у форматі Excel на основі різних параметрів запиту. Цей файл також обробляє логіку завантаження та пересилання фотографій, наданих користувачами та є збіркою допоміжних функцій дл ізвємодії з базою даних та обробки даних для бота. Визначає шлях для файлу Excel та файлу фотографій, а також списки назв стовпців, які використовуються для генерації звітів. Саме в цьому файлі прописано процедуру переформатування рядків дати, систему завантаження надісланої користувачем фотографії. Також в **db_utils.py** створена функція по надсиланню згенерованого файлу Excel адміністраторам. Операції з базами даних допоможуть додати новий запис співробітника до таблиці MyEmployees та новий спортивний запис до таблиці MyAthletics, пов'язуючи його з існуючим співробітником. Варто зауважити, що саме в цьому файлі є центральна функція для генерації запитів до бази даних на основі різних критеріїв.

Файл **states.py** є головним для інтерактивного характеру бота, визначаючи стани розмови, які ведуть користувачів через реєстрацію, подачу результатів та адміністративні запити. Це дозволяє боту запам'ятовувати контекст взаємодії користувача. Визначає стани кінцевого автомату станів (FSM), які використовуються фреймворком *aiogram* для керування потоками розмов в боті. Кожен клас представляє окремий контекст розмови, а його атрибути – це окремі кроки (стани) у цій розмові. *FormRegistration* визначає

стани для процесу реєстрації користувачів, направляючи користувача через введення його особистих та спортивних даних. *FormAthletics* визначає стани для запису спортивних результатів, запитуючи дисципліну, дату, час, дистанцію, результат та фотографію. *FormQuery* визначає стани для адміністративних запитів, дозволяючи адміністраторам вказувати критерії для пошуку в базі даних, включаючи загальну ініціацію запиту, конкретні поля, такі як прізвище, регіон, компанія, дисципліна та різні варіанти запитів за датою.

Загалом, тека Utils формує основу можливостей бота щодо збереження даних, звітності та структурованих розмов, абстрагуючи складну логіку бази даних та FSM від основних файлів обробників.

3.3.5 Структура. Тека Keyboards

Тека Keyboards містить модулі для створення та керування інтерактивними елементами користувацького інтерфейсу в боті. Ці елементи відіграють ключову роль у покращенні взаємодії з користувачем, надаючи йому чіткі варіанти дій.

Файл **builders.py** відповідає за клавіатури відповідей, які дозволяють користувачам вибирати з попередньо визначених варіантів у зручному форматі під полем введення повідомлення. Вони гнучко налаштовуються для різних сценаріїв взаємодії, наприклад, для вибору регіону або виду спорту. Основна функція приймає рядок та змінну кількість аргументів. Цей файл є центральним для створення інтерактивних кнопок, які з’являються під полем введення тексту в Telegram і спрямовують користувача через різні етапи взаємодії з ботом.

Файл **inline.py** призначений для визначення вбудованих клавіатур. Він імпортує потрібні бібліотеки для створення таких клавіатур. Файл використовується для надання користувачам можливості переходу за

зовнішнім посиланням безпосередньо з чату, наприклад, для завантаження медіафайлів.

Разом ці файли забезпечують візуальне та функціональне керування потоком користувача в боті, надаючи йому інтуїтивно зрозумілі способи введення даних та взаємодії з системою. Вони є невід'ємною частиною для побудови ефективного та зручного для користувача інтерфейсу Telegram-бота.

Висновки до розділу 3

У Розділі 3 було детально розглянуто проектування та реалізацію інтелектуальної системи для фіксації спортивних досягнень співробітників компанії. Архітектура системи передбачає серверну та клієнтську частини, причому серверна частина реалізована на основі бази даних, а клієнтська поділяється на користувацьку та адміністративну.

Користувацька частина дозволяє незареєстрованим користувачам пройти обов'язкову процедуру реєстрації, що включає введення повних особистих та контактних даних, а також інформації про спортивні вподобання. Зареєстровані ж користувачі отримують можливість фіксувати свої спортивні результати з бігу, їзди на велосипеді та дуатлону, зазначаючи дату, час початку, дисципліну, тривалість та подолану дистанцію. Важливою особливістю є вимога додавати фотопідтвердження для верифікації даних. Система також передбачає валідацію введених даних, надаючи користувачеві відповідні повідомлення у разі некоректного введення.

Адміністративна частина призначена для відповідальних співробітників компанії, які мають доступ до розширеного функціоналу. Адміністратори можуть створювати різноманітні запити до бази даних для систематизації та фільтрації інформації про спортивні досягнення. Серед критеріїв запитів доступні повна база учасників, фільтрація за прізвищем, регіоном (з переліком конкретних регіонів), компанією, видом спорту, а також за точною датою або певним діапазоном часу. Результати запитів формуються у зручному для аналізу форматі.

У процесі вибору алгоритмів та методів обробки даних були обґрунтовані ключові технологічні рішення:

База даних **PostgreSQL** обрана завдяки її надійності (ACID-властивості), підтримці широкого спектру типів даних, можливості розширення, потужним інструментам аналізу (розширений SQL, віконні функції), масштабованості та економічній ефективності (безкоштовність та активна спільнота).

Мова програмування **Python** вибрана за її простоту синтаксису, читабельність коду, широку екосистему бібліотек та фреймворків (особливо для інтелектуальних функцій, аналізу даних, візуалізації та машинного навчання), кросплатформність та активну спільноту підтримки.

Середовище розробки **PyCharm Professional** було обрано як оптимальний інструмент, що забезпечує інтелектуальну допомогу в кодуванні, потужну перевірку коду, можливості рефакторингу, глибоку інтеграцію з базами даних (зокрема PostgreSQL), підтримку наукових обчислень (Jupyter Notebook, бібліотеки) та ефективні інструменти налагодження та управління версіями.

Формування відповіді для адміністраторів реалізовано у форматі файлів **Microsoft Excel**, що є зручним та звичним для офісних працівників, не вимагає додаткового навчання та дозволяє легко поширювати та аналізувати дані.

Інтерфейс користувача системи реалізовано у вигляді Telegram-бота, який автоматично визначає статус клієнта (адміністратор чи користувач) за його ідентифікаційним номером. Розділ також детально описує структуру програмного коду, розділену на логічні теки:

Handlers: Містить обробники для взаємодії з користувачем, керуючи процесами реєстрації, фіксації спортивних результатів та адміністративними запитами (user_commands.py, form_admin_query.py, form_registration.py, form_athletics.py).

Utils: Включає допоміжні компоненти для керування базою даних (db.py, db_utils.py), визначення команд бота (commands.py) та станів кінцевих автоматів (states.py).

Keyboards: Відповідає за створення та керування інтерактивними елементами інтерфейсу, такими як клавіатури відповідей (builders.py) та вбудовані клавіатури (inline.py).

Загалом, описаний розділ демонструє комплексний підхід до проектування та реалізації інтелектуальної системи, яка поєднує сучасні технології для ефективного керування спортивними даними, забезпечення

зручного користувацького досвіду та потужних адміністративних можливостей.

РОЗДІЛ 4

ТЕСТУВАННЯ ТА ОЦІНКА ПРАЦЕЗДАТНОСТІ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ

4.1 Сценарії використання системи

Коли робота повноцінно готова, код написаний, всі перевірки пройдені і є впевненість в коректному виконанні тих чи інших функцій, то потрібно розглянути інструкцію використання бота для користувача, для адміністратора та для керівників компанії.

4.1.1 Для керівників

Є багато компаній, де керівники мають бажання об’єднувати своїх співробітників за допомогою різноманітних спільних заходів. Часто це відбувається у вигляді виїзних корпоративів, на яких можна покращувати взаємодію один з одним, шляхом, так званого, тімбілдінга з цікавими активностями.

Проте корпоративи не можуть відбуватись щодня, адже це буде перешкоджати основній роботі підприємства. Тому задля можливості об’єднувати співробітників з різних філіалів та відділів компанії по всій країні, було прийнято рішення створити між працівниками певну конкуренцію. Водночас, цю конкуренцію важливо зробити по-справжньому корисною. Для цього було обрано спорт.

Завдяки спортивній конкуренції співробітники різних компаній зможуть розділитись на групи, згідно зі своєю спортивною підготовкою. А щоб у всіх учасників спортивних активностей була мотивація до цієї конкуренції, керівники компаній можуть ввести систему фінансових нагород. Наприклад, ті працівники, які зможуть подолати найбільшу дистанцію в кожній із спортивних дисциплін, отримають премію в розмірі, який перед початком

змагань оголосить керівник компанії. Це спонуватиме працівників підтримувати свою фізичну форму та вести здоровий спосіб життя. А найголовніше – це об’єднає багатьох співробітників в одне ціле в ході здорової конкуренції між собою.

Керівникам необхідно обрати відповідальну особу, яка час від часу буде підбивати підсумки та передавати всю інформацію про лідерів керіникам, які в свою чергу, на основі наданих результатів, виписуватимуть фінансові винагороди.

Це може відбуватись з будь-якою періодичністю. Все залежить від рішення керівника тієї чи іншої компанії. Може бути раз в рік, раз в пів року, раз в тиждень тощо. Проте найоптимальнішим варіантом періодичності фінансових нагород та підбивання підсумків є місяць. Мабуть, найкращим рішенням для керівників будуть щомісячні премії найкращим учасникам та лідерам рейтингу!

Керівники виступають ініціаторами впровадження системи, що має на меті об'єднання співробітників з різних філіалів через здорову спортивну конкуренцію. Вони визначають систему фінансових винагород для стимулювання учасників і встановлюють періодичність підбиття підсумків (оптимально — щомісячно), покладаючи відповідальність за збір даних на адміністратора (якого попепредньо призначають). Цей підхід сприяє не лише фізичному розвитку, а й покращенню взаємодії та тімбілдінгу в масштабах компанії.

4.1.2 Для адміністраторів

Адміністратором є відповідальна особа, визначена керівником. В обов’язки адміністратора входить щомісячна або щотижнева перевірка усіх рейтингів, аналіз результатів по різних критеріях та передавання всієї важливої інформації керівникам.

Враховуючи, що ідея покращення спортивної форми та об’єднання співробітників може досягти загальнодержавного масштабу, то і учасників буде дуже багато, а тому потрібне детальне створення рейтингів та статистики по всім можливим критеріям пошуку.

Адміністратор є ключовою відповідальною особою, яка призначається керівництвом. Його завдання полягає у регулярній (щомісячній або щотижневій) перевірці та аналізі всіх рейтингів і статистики за різними критеріями. З огляду на потенційний загальнодержавний масштаб проєкту, що може призвести до значної кількості учасників, роль адміністратора в детальному формуванні рейтингів та наданні інформації керівництву є критично важливою.

4.1.3 Для користувачів

Завдання користувача, мабуть, найлегше. Його задача при першому вході в інтелектуальну систему зареєструватися, ввівши всі необхідні дані. Після кожного заняття спортом, користувач повинен заповнюваит в боті всю інформацію про своє заняття.

Після чого, слідкувати за оновленнями статистики та рейтингу, відслідковувати свою позицію, порівняно з іншими працівниками та завжди намагатися бути на вершині будь-якого рейтингу задля отримання найкращих премій. Конкурувати зі своїми колегами, тим самим об’єднуватися завдяки спільному спортивному інтересу. Декотрі співробітники можуть об’єднуватися в групи та займатися спортом разом. Спільний біг або катання на велосипеді допоможе не лише підтримувати фізичну форму, а ще й завдяки спілкуванню, в ході спортивних активностей заводити нових друзів серед колег.

Користувач має найпростіший, але фундаментальний обов'язок: первинна реєстрація в системі та регулярне внесення інформації про свої спортивні заняття. Мотивація користувача підтримується можливістю відстежувати

свою позицію в рейтингу, конкурувати з колегами та прагнути до лідерства для отримання премій.

Система також заохочує спільні заняття спортом, що додатково сприяє соціалізації та об'єднанню працівників в одне ціле та створення комформного клімату в колективі.

Так як і в кожній компанії, незалежно від сфери, на якій спеціалізується підприємство, кожен працівник намагається бути краще за інших. Хтось хоче таким бути задля фінансової вигоди у вигляді додаткових матеріальних виплат, хтось із співробітників має «синдром лідера» і йому для власного его важливо бути завжди та всюди першим, при чому для таких людей абсолютно не має великого значення сфера в якій треба бути кращим. Але всіх цих співробітників компанії об’єднує одне – жага до першого місця в будь-якому питанні.

Спортивна конкуренція завжди була великим та дуже дієвим стимулом для кожної людини аби покаазати свою силу, витривалість та багато інших фізичних та психологічних якостей. Водночас із тим, керівники будуть бачити не лише націленість кожного свого підлеглого на перемогу. А й оцінювати вміння ніколи не зупинятися та йти завжди до кінця.

Як відомо, будь-який роботодавець неабияк цінує цілеспрямованих та зацікавлених працівників. А спорт – це та сфера, яка цікавить більшість населення всієї планети Земля, незалежно від віку, статусу, місця роботи та посади.

4.2 Функціональне тестування

4.2.1 RegExp в роботі

Інтелектуальна система створена таким чином, що користувач може вводити лише коректні дані. Точніше вводити можна будь-які, але система пропустить користувача на наступне питання та зробить запис в базу даних

лише у випадку введення коректних даних. Це зроблено завдяки регулярним виразам.

Регулярні вирази (RegExp) — це спеціальні послідовності символів, що використовуються для пошуку та маніпулювання текстом на основі складних шаблонів. Вони дозволяють ефективно знаходити, замінювати та перевіряти відповідності у рядках.

Регулярні вирази у Python є потужним інструментом для обробки та маніпулювання текстом. Вони надають гнучкий і ефективний спосіб пошуку, зіставлення, вилучення та заміни послідовностей символів на основі визначених шаблонів.

Вбудований модуль **re** у Python є основним інтерфейсом для роботи з регулярними виразами. Він дозволяє програмістам створювати складні шаблони, які можуть відповідати різним варіаціям тексту, а не лише фіксованим рядкам.

Основна концепція регулярних виразів полягає у використанні спеціальних метасимволів та операторів у шаблоні. Ці метасимволи надають регулярним виразам їхню велику потужність, дозволяючи задавати такі важливі речі, як повторення символів, набори символів, позиції в рядку, а також групувати частини шаблону для вилучення в подальшому або маніпуляції.

Важливою особливістю є можливість компіляції регулярних виразів. Якщо один і той же шаблон використовується багаторазово, його можна скомпілювати в об'єкт регулярного виразу. Це покращує продуктивність, оскільки Python розбирає шаблон тільки один раз, а не при кожному виконанні.

Загалом, регулярні вирази в Python є невід'ємною частиною інструментарію для будь-кого, хто займається обробкою тексту, аналізом даних або валідацією вхідної інформації. від простого пошуку до складного парсингу даних.

Регулярні вирази в Python є потужним інструментом для роботи з текстом, який дозволяє знаходити, зіставляти та маніпулювати рядками на основі складних шаблонів. Ця функціональність реалізована через вбудований модуль re.

Основна ідея полягає в тому, що програміст визначаєте шаблон (регулярний вираз), який описує послідовність символів, яку він хоче знайти. Цей шаблон може включати як звичайні символи (які відповідають самі собі), так і спеціальні "метасимволи", що мають особливе значення. Метасимволи надають регулярним виразам їхню гнучкість, дозволяючи вказувати повторення символів, діапазони символів, початок/кінець рядка, або навіть групи символів.

Функціональне тестування підтверджує надійність системи та її здатність обробляти коректні дані, що забезпечується завдяки інтеграції регулярних виразів (RegExp). RegEx у Python, використовуючи вбудований модуль re, є основним інструментом для валідації вхідних даних від користувачів. Це гарантує, що лише коректні дані будуть записані в базу даних, тоді як некоректні вводи відхиляються, тим самим забезпечуючи цілісність інформації. Використання регулярних виразів дозволяє створювати гнучкі шаблони для перевірки відповідності тексту, а можливість компіляції шаблонів оптимізує продуктивність системи.

4.2.2 Викладання бота на хостинг

Початкове тестування програмного коду проводилось в програмі PyCharm Professional. Проте PyCharm можна використовувати лише для попереднього тестування. Для того, щоб система працювала весь час, незалежно від того чи увімкнений ПЕОМ на якому запускається програма, потрібноо придбати хостинг у провайдера. Дану інформаційну систему було викладено на платформу хостинг-провайдера TheHost.

Даний хостинг-провайдер пропонує розміщення Телеграм-ботів на віртуальному хостингу з cPanel або на VPS/VDS для більш складних та рідкісних випадків.

Для початку потрібно обрати тип хостингу. VPS/VDS – потужний та гнучкий варіант, який неодмінно знадобиться, якщо бот потребує постійного фонового процесу, виконує ресурсоємні завдання, має велику кількість одночасних користувачів, потребує root-доступу або специфічних налаштувань даного сервера. Також існує інший варіант – віртуального хостингу – це наспростіший і найдешевший варіант, який підходить для невеликих і простих ботів, які не вимагають постійного фонового процесу та не є ресурсоємними.

Після вибору типу хостингу необхідно підотувати бот. Для цього отримаємо API-токен та виберемо метод взаємодії з Telegram API. Тут доведеться обирати між Webhook та Long Polling. Для VPS зазвичай використовується Long Polling. В такій ситуації бот постійно запитуватиме Telegram API про нові оновлення. Це вимагає постійно запущеного процесу. На віртуальному хостингу це може бути проблемою, адже фонові процеси часто обмежуться або примусово завершуються.

Наступним кроком є розміщення файлів бота. Використовується FTP-клієнт для завантаження файлів бота на сервер. Бот використовує багато бібліотек, тому необхідно встановити залежності на хостингу вручну за допомогою pip після підключення через SSH. Такі дії зможе зробити навіть недосвідчений користувач, завдяки влеикій кількості літератури та вебсайтів з структурою, праавилами, вимогами та підказками до використання даної системи.

Потрубно налаштувати та запустити хостинг. На VPS/VDS можна використовувати screen, tmux, systemd або supervisor для запуску бота у фоновому режимі та його автоматичного перезапуску у разі збою. Якщо ж це відбувається на віртуальному хостингу, то буде багато складнощів. Деякі

хостери можуть мати обмеження на час виконання скриптів або використання ресурсів.

Враховуючи використання бази даних, треба переконатися, що вона налаштована на хостингу і що бот має до неї доступ. Перевірити логи сервера та логи бота для виявлення та виправлення помилок. А SSH-доступ дозволить виконувати команди, перевіряти статус процесів та переглядати будь-які логи за вашим бажанням.

Загалом, людям, які хочуть працювати з цією системою, треба ознайомитись з офіційною документацією TheHost та статтями бази знань, оскільки вони можуть мати конкретні інструкції для розгортання Telegram-ботів.

4.2.3 Визначення обмежень та перспектив розвитку

На даний момент явних обмежень немає, проте є вимоги до програмного забезпечення.

Для серверу повинна бути операційна система MS Windows 10/8/7/Vista/2003/XP або Ubuntu, а версія Python 2.4 або вище, PyPy або IronPython.

Користувач, в свою чергу, повинен встановити застосунок Telegram на будь-яку платформу. Якщо це портативний комп’ютер або ноутбук, то операційна система повинна бути MS Windows 10/8/7/Vista/2003/XP;

Також є вимоги до апаратної частини: комп’ютер на базі процесора Intel Pentium 166 і вище з оперативною пам’яттю не менше 2Гбайт. Мінімальна роздільна здатність екрану 1024x768 та вільний простір жорсткого диску не менше 300 Мбайт з підключенням до Інтернету.

Розвиток системи не має обмежень. Якщо ідея буде масово запроваджуватися для різних компаній та буде відслідковуватись зацікавленість користувачів у цій системі, то подібний алгорит з преміювання

співробітників за спортивні досягнення може дійти навіть до міжнародного рівня.

Висновки до розділу 4

Розділ присвячений всебічному тестуванню та оцінці працездатності розробленої інтелектуальної системи, що є Telegram-ботом, спрямованим на підвищення фізичної активності та об'єднання співробітників компаній. Детально розглянуто сценарії використання системи для трьох ключових груп: керівників, адміністраторів та користувачів, а також ключові аспекти функціонального тестування, включаючи застосування регулярних виразів та процес розгортання бота на хостингу.

Для керівників визначено їхню роль як ініціаторів впровадження системи, що стимулює тімбілдинг та покращення взаємодії між співробітниками з різних філіалів. Адміністратори, призначені керівництвом, виконують роль відповідальних осіб за регулярну перевірку, аналіз рейтингів та статистики, а також передачу основної інформації керівникам. Користувачі мають найпростіший, але фундаментальний обов’язок – первинна реєстрація та регулярне внесення даних про свої спортивні занятті.

Функціональне тестування підтвердило надійність системи та її здатність обробляти коректні дані. Це забезпечується завдяки інтеграції регулярних виразів, які є основним інструментом для ваоідаціх вхідних даних від користувачів, гарантуючи цілісність інформації в базі даних.

Також було детально описано розгортання боту на хостинку, зокрема на платформі TheHost. Бцло розглянуто два типи хостингу (віртуальний хостинг та VPS/VDS) та їхні особливості, методи взаємодії з Telegram API (Webhook та Long Polling), а також необхідні кроки для розміщення файлів бота та налаштування залежностей

Окрему увагу було приділено визначенню обмежень та перспектив розвитку системи. Наразі відсутні явні обмеження, проте існують певні вимоги до програмного забезпечення та до апаратної частини для користувачів системи.

Загалом, цей розділ демонструє комплексний підхід до тестування та оцінки інтелектуальної системи, підтверджуючи її функціональність, надійність та готовність до практичного застосування.

СПИСОК ЛІТЕРАТУРИ

1. Історія ШІ (Вікіпедія) [Електронний ресурс] Посилання на ресурс: https://uk.wikipedia.org/wiki/%D0%86%D1%81%D1%82%D0%BE%D1%80%D1%96%D1%8F_%D1%88%D1%82%D1%83%D1%87%D0%BD%D0%BE%D0%B3%D0%BE_%D1%96%D0%BD%D1%82%D0%B5%D0%BB%D0%B5%D0%BA%D1%82%D1%83 (дата звернення: 19.05.2025)

2. Майбутнє ШІ: Тенденції та перспективи розвитку (Паяльня) [Електронний ресурс] Посилання на ресурс: https://payalnya.com.ua/uk/blog-uk/maybutnie-shtuchnogo-intelektu-tendencii-ta-perspektivi-rozvitkuuk/?srsltid=AfmBOoqKkIAvjXLqfnbm9o_moAkUrk9gzXJErM61gKZU7B9Yqn9QbR_v (дата звернення: 19.05.2025)

3. Топ найкращих застосунків для бігу (SPEKA.media) [Електронний ресурс] Посилання на ресурс: https://speka.media/sist-zastosunkiv-dlya-bigu-yaki-dopomozut-efektivno-trenuvatisya-ta-ne-vtratiti-progres-py66kv (дата звернення: 20.05.2025)

4. Про що має знати розробник Телеграм-ботів (Habr) [Електронний ресурс] Посилання на ресурс: https://habr.com/ru/articles/543676/ (дата звернення: 22.05.2025)

5. Telegram (Вікіпедія) [Електронний ресурс] Посилання на ресурс: https://uk.wikipedia.org/wiki/Telegram (дата звернення: 22.05.2025)

6. Типи баз даних: особливості, відмінності та приклади (DOU.ua) [Електронний ресурс] Посилання на ресурс: https://dou.ua/lenta/articles/types-of-databases/ (дата звернення: 23.05.2025)

7. Що таке SQL та бази даних? (Acode.com.ua) [Електронний ресурс] Посилання на ресурс: https://acode.com.ua/sql-intro/ (дата звернення: 23.05.2025)

8. PostgreSQL: The world’s most advanced open source database (PostgreSQL) [Електронний ресурс] Посилання на ресурс: https://www.postgresql.org/ (дата звернення: 23.05.2025)

9. Чим PostgreSQL краще інших SQL баз даних? (Habr) [Електронний ресурс] Посилання на ресурс: https://habr.com/ru/articles/282764/ (дата звернення: 23.05.2025)

10. Logic Theorist – Knowledge and References (Logic Theorist) [Електронний ресурс] Посилання на ресурс: https://taylorandfrancis.com/knowledge/Engineeri

ng_and_technology/Artificial_intelligence/Logic_Theorist/ (дата звернення: 19.05.2025)

11. ELIZA: The First Step in Human-Computer Interaction Through Natural Language Processing (GeeksforGeeks) [Електронний ресурс] Посилання на ресурс: https://www.geeksforgeeks.org/eliza-the-first-step-in-human-computer-interaction-through-natural-language-processing/ (дата звернення: 19.05.2025)

12. Backend.AI Documentation (Backend.AI) [Електронний ресурс] Посилання на ресурс: https://docs.backend.ai/en/latest/ (дата звернення: 25.05.2025)

13. IDLE – Python editor and shell (Python Documentation) [Електронний ресурс] Посилання на ресурс: https://docs.python.org/3/library/idle.html (дата звернення: 26.05.2025)

14. PyCharm: The only Python IDE you need (JetBrains) [Електронний ресурс] Посилання на ресурс: https://www.jetbrains.com/pycharm/ (дата звернення: 26.05.2025)

15. Порівняння версій PyCharm: Community Edition vs Profesional Edition (Hexlet.io) [Електронний ресурс] Посилання на ресурс: https://ru.hexlet.io/blog/posts/sravnenie-versiy-pycharm-community-edition-vs-professional-edition (дата звернення: 26.05.2025)

16. Our Documentation (Python.org) [Електронний ресурс] Посилання на ресурс: https://www.python.org/doc/ (дата звернення: 26.05.2025)

17. Welcome to Python.org (Python.org) [Електронний ресурс] Посилання на ресурс: https://www.python.org/doc/ (дата звернення: 26.05.2025)

18. Ukrainian translation of the Python Documentation (GitHub) [Електронний ресурс] Посилання на ресурс: https://github.com/python/python-docs-uk (дата звернення: 26.05.2025)

19. re – Regular expression operations (Python.org) [Електронний ресурс] Посилання на ресурс: https://docs.python.org/uk/3.13/library/re.html (дата звернення: 27.05.2025)

20. Хостингова Компанія TheHost (TheHost.ua) [Електронний ресурс] Посилання на ресурс: https://thehost.ua/ua/hosting/plans (дата звернення: 29.05.2025)

21. MySQL (mysql.com) [Електронний ресурс] Посилання на ресурс: https://www.mysql.com/ (дата звернення: 22.05.2025)

22. Building Telegram Bot with Python: A Comprehensive Guide (Medium) [Електронний ресурс] Посилання на ресурс: https://medium.com/@moraneus/building-telegram-bot-with-python-telegram-bot-a-comprehensive-guide-7e33f014dc79 (дата звернення: 05.05.2025)

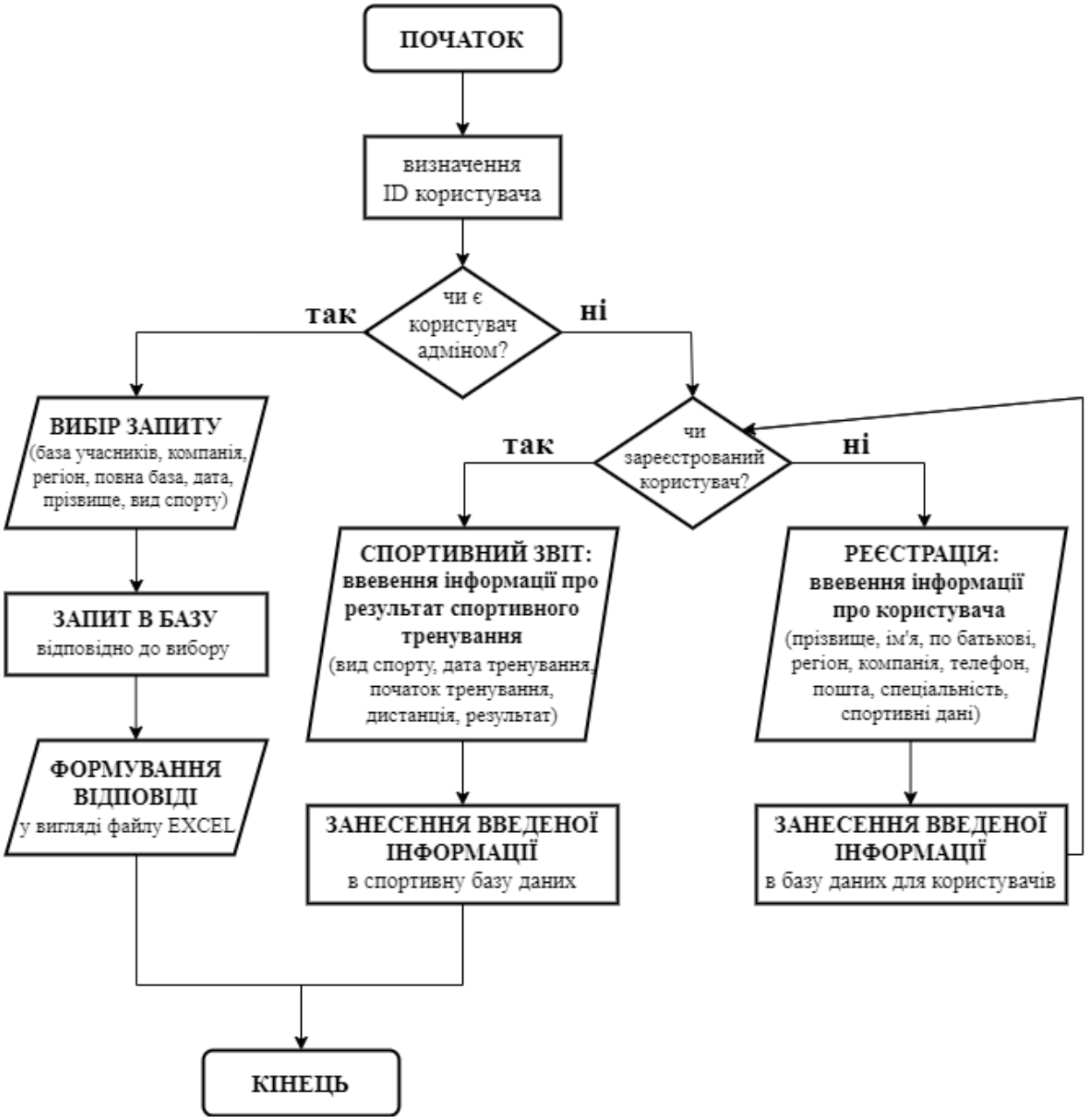
ДОДАТОК 1

Інтелектуальна система для відслідковування спортивних досягнень працівників компанії

Алгоритм роботи системи

ІАЛЦ.467200.004 Д1

Аркушів 1



Зм.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467200.004 Д1		
Розроб.		Потапенко М.В.			Інтелектуальна система для відслідковування спортивних досягнень працівників компанії Алгоритм роботи системи	Літ.	Арк. 1 / Аркушів 1
Перевір.		Шульга М.В.					
Н.контр.		Пономаренко А.М				КПІ ім. Ігоря Сікорського ФІОТ, ІМ-13	
Затверд.							

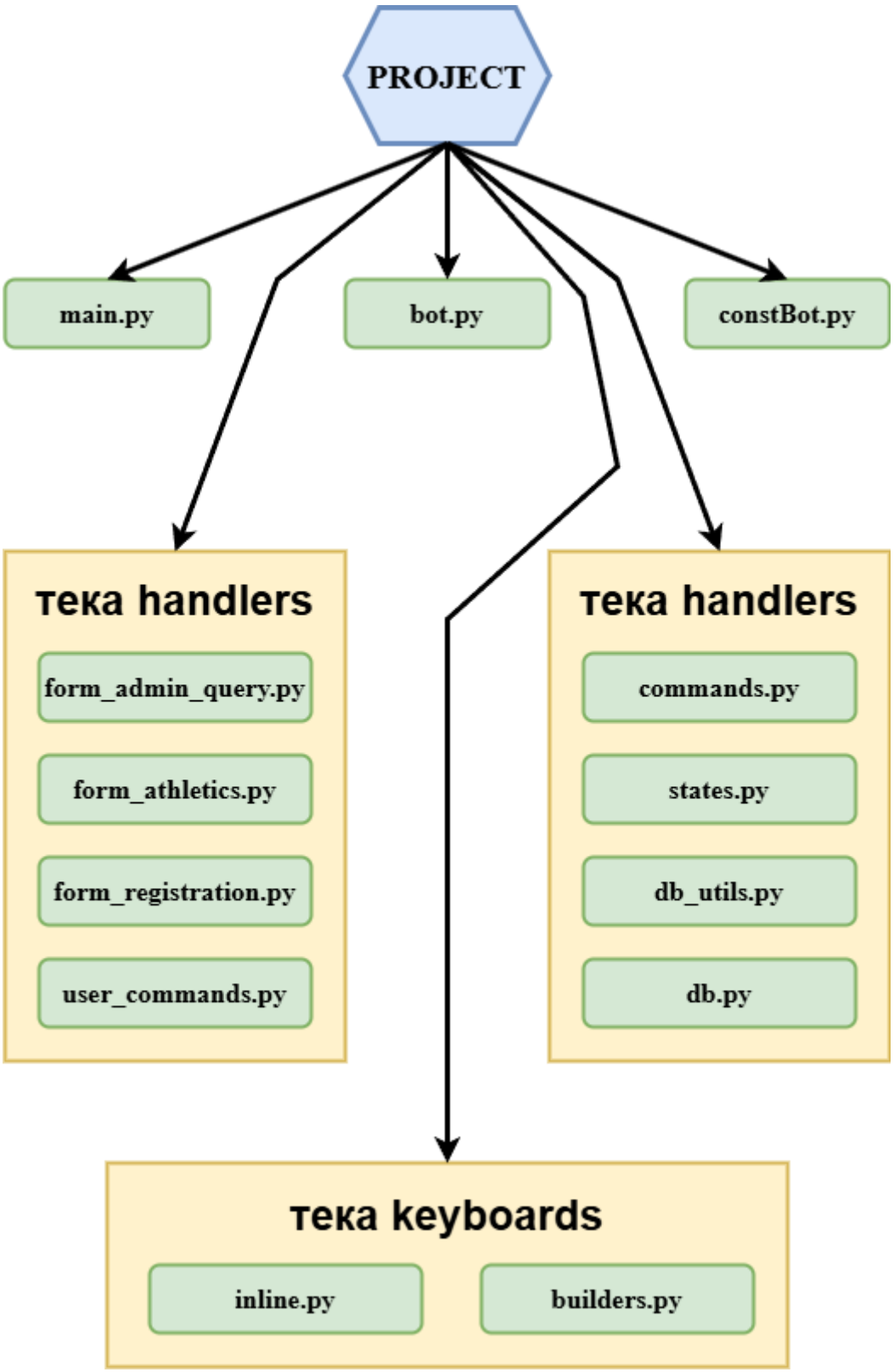
ДОДАТОК 2

Інтелектуальна система для відслідковування спортивних досягнень працівників компанії

Загальна структура побудови коду

ІАЛЦ.467200.005 Д2

Аркушів 1



					ІАЛЦ.467200.005 Д2		
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розроб.		Потапенко М.В.			*Інтелектуальна система для відслідковування спортивних досягнень працівників компанії* Загальна структура побудови коду	*Літ.*	*Арк.* / *Аркушів*
Перевір.		Шульга М.В.					1 / 1
Н.контр.		Пономаренко А.М				***КПІ ім. Ігоря Сікорського ФІОТ, ІМ-13***	
Затверд.							

ДОДАТОК 3

Інтелектуальна система для відслідковування
спортивних досягнень працівників компанії

Структурна система бази даних

ІАЛЦ.467200.006 Д3

Аркушів 1

MyEmployees

НАЗВА ПАРАМЕТРУ	ТИП ДАНИХ
ID	*Integer*
surname	*String*
name	*String*
father_name	*String*
region	*String*
company	*String*
speciality	*String*
phone	*String*
email	*String*
telegramID	*String*
base_sport	*String*
sport	*String*
category	*String*

MyAthletics

НАЗВА ПАРАМЕТРУ	ТИП ДАНИХ
ID	*Integer*
Date	*Date*
Time	*Time*
Discipline	*String*
Distance	*Integer*
Result	*Time*
employee_id	*Integer*

Зм.	Арк.	№ докум.	Підпис	Дата	***ІАЛЦ.467200.006 ДЗ***		
Розроб.		Потапенко М.В.			*Інтелектуальна система для відслідковування спортивних досягнень працівників компанії* Структурна схема бази даних	*Літ.*	*Арк.* 1 / *Аркушів* 1
Перевір.		Шульга М.В.					
Н.контр.		Пономаренко А.М				***КПІ ім. Ігоря Сікорського ФІОТ, ІМ-13***	
Затверд.							

ДОДАТОК 4

Інтелектуальна система для відслідковування спортивних досягнень працівників компанії

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 16

ЗМІСТ

main.py

```
from aiogram import Bot
from aiogram.enums import ParseMode
from constBot import TOKEN_SPORT_AGRO_BOT, ID_VIT

TOKEN = TOKEN_SPORT_AGRO_BOT
bot = Bot(TOKEN, parse_mode=ParseMode.HTML)
ID_ADMIN = ID_VIT
ID_SUPERADMIN = ID_VIT
```

bot.py

```
import asyncio

from aiogram import Dispatcher
from main import bot


from handlers import (
    user_commands,
    form_registration,
    form_athletics,
    form_admin_query
)

dp = Dispatcher()


dp.include_routers(
    user_commands.router,
    form_registration.router,
    form_athletics.router,
    form_admin_query.router
  )


async def main():
    await bot.delete_webhook(drop_pending_updates=True)
    await dp.start_polling(bot)


if __name__ == '__main__':
    asyncio.run(main())
```

constBot.py

```
TOKEN_SPORT_AGRO_BOT = '7092985511:AAGpPzh_A50n3E8dxT6v5GcpiaOgsqtvIos'
ID_MISHA = 5862438426
ID_VIT = 564716071
```

handlers/form_admin_query.py

```
from aiogram import Router, F
from aiogram.types import Message
from aiogram.fsm.context import FSMContext
```

```
from utils.states import FormQuery
from utils.db_utils import get_base_query, get_base_query_empl
from keyboards.builders import profile


router = Router()


@router.message(F.text.lower() == 'адміністрування')
async def fill_profile(message: Message, state: FSMContext):
    await state.set_state(FormQuery.start)
    await message.answer("По якому критерію бажаєте зробити запит?",
            reply_markup=profile(["База учасників", "Повна база", "Прізвище",
                                  "Регіон", "Компанія", "Вид спорту","Дата"],
                                  2, 1, 2, 2))


@router.message(FormQuery.start, F.text.casefold().in_(["база учасників"]))
async def form_query_employees(message: Message, state: FSMContext):
    await state.update_data(start=message.text)
    await get_base_query_empl(message)
    await message.answer("База учасників створена")


@router.message(FormQuery.start, F.text.casefold().in_(["повна база"]))
async def form_query_all(message: Message, state: FSMContext):
    await state.update_data(start=message.text)
    await get_base_query(message.text, message, "all")
    await message.answer("Повна база створена")


@router.message(FormQuery.start, F.text.casefold().in_(["прізвище"]))
async def form_query_surname(message: Message, state: FSMContext):
    await state.update_data(start=message.text)
    await state.set_state(FormQuery.surname)
    await message.answer("Введіть прізвище")


@router.message(FormQuery.surname)
async def form_surname(message: Message, state: FSMContext):
    await state.update_data(surname=message.text)
    if await get_base_query(message.text, message, "surname") is True:
        await message.answer("Запит по Прізвищу створено")
    else:
        await message.answer("Даних, що відповідають запиту не знайдено")


@router.message(FormQuery.start, F.text.casefold().in_(["регіон"]))
async def form_query_region(message: Message, state: FSMContext):
    await state.update_data(start=message.text)
    await state.set_state(FormQuery.region)
    await message.answer("Оберіть регіон",
                         reply_markup=profile(["Київщина", "Полтавщина",
                                               "Сумщина", "Чернігівщина",
                                               "Волинь", "Житомирщина",
                                               "Харківщина", "Тернопільщина",
                                               "Хмельниччина", "Вінниччина",
                                               "Черкащина", "Рівненщина",
                                               "Львівщина"],
                                               1, 2, 2, 2, 2, 2, 2))


@router.message(FormQuery.region, F.text.casefold().in_(["київщина",
```

```
                                                "полтавщина", "сумщина",
                                                "чернігівщина", "волинь",
                                                "житомирщина", "харківщина",
                                                "тернопільщина", "хмельниччина",
                                                "вінниччина", "черкащина",
                                                "рівненщина", "львівщина"]))

async def form_region(message: Message, state: FSMContext):
    await state.update_data(region=message.text)
    if await get_base_query(message.text, message, "region") is True:
        await message.answer("Запит по Регіону створено")
    else:
        await message.answer("Даних, що відповідають запиту не знайдено")


@router.message(FormQuery.region)
async def incorrect_form_date(message: Message, state: FSMContext):
    await message.answer("Оберіть з наданих варіантів")


@router.message(FormQuery.start, F.text.casefold().in_(["компанія"]))
async def form_query_company(message: Message, state: FSMContext):
    await state.update_data(start=message.text)
    await state.set_state(FormQuery.company)
    await message.answer("Введіть компанію")


@router.message(FormQuery.company)
async def form_company(message: Message, state: FSMContext):
    await state.update_data(company=message.text)
    if await get_base_query(message.text, message, "company") is True:
        await message.answer("Запит по Компанії створено")
    else:
        await message.answer("Даних, що відповідають запиту не знайдено")


@router.message(FormQuery.start, F.text.casefold().in_(["вид спорту"]))
async def form_query_discipline(message: Message, state: FSMContext):
    await state.update_data(start=message.text)
    await state.set_state(FormQuery.discipline)
    await message.answer("Оберіть вид спорту",
                         reply_markup=profile(["БІГ", "ВЕЛО", "ДУАТЛОН"],2))


@router.message(FormQuery.discipline, F.text.casefold().in_(["біг", "вело",
"дуатлон"]))
async def form_discipline(message: Message, state: FSMContext):
    await state.update_data(discipline=message.text)
    if await get_base_query(message.text, message,"discipline") is True:
        await message.answer("Запит по Виду спорту створено")
    else:
        await message.answer("Даних, що відповідають запиту не знайдено")


@router.message(FormQuery.discipline)
async def incorrect_form_date(message: Message, state: FSMContext):
    await message.answer("Оберіть з наданих варіантів")


@router.message(FormQuery.start, F.text.casefold().in_(["дата"]))
async def form_query_date(message: Message, state: FSMContext):
    await state.update_data(start=message.text)
    await state.set_state(FormQuery.date)
    await message.answer("Виберіть потрібний варіант",
```

```
                        reply_markup=profile(["Точна дата", "Від __ і до
сьогодні", "Діапазон"], 1, 1))


@router.message(FormQuery.date, F.text.casefold().in_(["точна дата", "від __
і до сьогодні"]))
async def form_date(message: Message, state: FSMContext):
    await state.update_data(date=message.text)
    await state.set_state(FormQuery.one_date)
    await message.answer("Введіть дату")


@router.message(FormQuery.one_date, F.text.regexp(r"^[0-3][0-9].[0-1][0-
9].24$"))
async def form_one_date(message: Message, state: FSMContext):
    await state.update_data(one_date=message.text)
    data = await state.get_data()
    param_date = data.get("date")

    if param_date == "Точна дата":
        if await get_base_query(message.text, message,"one_date") is True:
            await message.answer("Запит по Точній даті створено")
        else:
            await message.answer("Даних, що відповідають запиту не знайдено")
    else:
        if await get_base_query(message.text, message,"old_date") is True:
            await message.answer("Запит від заданої дати створено")
        else:
            await message.answer("Даних, що відповідають запиту не знайдено")
    await state.clear()


@router.message(FormQuery.one_date)
async def incorrect_form_one_date(message: Message, state: FSMContext):
    await message.answer("Введіть дату в форматі XX.XX.24")


@router.message(FormQuery.date, F.text.casefold().in_(["діапазон"]))
async def form_date_two(message: Message, state: FSMContext):
    await state.update_data(date=message.text)
    await state.set_state(FormQuery.two_date_start)
    await message.answer("Введіть дату від якої включно створити запит")


@router.message(FormQuery.two_date_start, F.text.regexp(r"^[0-3][0-9].[0-
1][0-9].24$"))
async def form_date_two_start(message: Message, state: FSMContext):
    await state.update_data(two_date_start=message.text)
    await state.set_state(FormQuery.two_date_end)
    await message.answer("Введіть дату до якої включно створити запит")


@router.message(FormQuery.two_date_end, F.text.regexp(r"^[0-3][0-9].[0-1][0-
9].24$"))
async def form_date_two_end(message: Message, state: FSMContext):
    await state.update_data(two_date_end=message.text)
    data = await state.get_data()
    two_date_start = data.get("two_date_start")
    if await get_base_query(two_date_start, message, "two_date",
message.text) is True:
        await message.answer("Запит по заданому періоду створено")
    else:
        await message.answer("Даних, що відповідають запиту не знайдено")
```

```
@router.message(FormQuery.two_date_start)
async def incorrect_form_two_date(message: Message, state: FSMContext):
    await message.answer("Введіть дату в форматі XX.XX.24")


@router.message(FormQuery.two_date_end)
async def incorrect_form_two_date2(message: Message, state: FSMContext):
    await message.answer("Введіть дату в форматі XX.XX.24")
```

handlers/form_athletics.py

```
from aiogram import Router, F
from aiogram.types import Message
from aiogram.fsm.context import FSMContext
from datetime import datetime
from main import bot, ID_ADMIN, ID_SUPERADMIN
from utils.states import FormAthletics
from utils.db_utils import add_base_athl, get_base_surname_name, get_photo
from keyboards.builders import profile
from keyboards.inline import video_photo_kb

router = Router()


@router.message(F.text.lower() == 'фіксація результатів')
async def fill_profile(message: Message, state: FSMContext):
    await state.set_state(FormAthletics.discipline)
    await message.answer("Доброго дня! \nВи позаймались спортом!\n"
                         "Давайте зафіксуємо результат Вашого тренування\n"
                         "Введіть яким саме спортом Ви займались?",
                         reply_markup=profile(["БІГ", "ВЕЛО", "ДУАТЛОН"], 3))


@router.message(FormAthletics.discipline, F.text.casefold().in_(["біг",
"вело", "дуатлон"]))
async def form_discipline(message: Message, state: FSMContext):
    await state.update_data(discipline=message.text)
    await state.set_state(FormAthletics.date)
    current_date = '{:%d.%m.%y}'.format(datetime.now())
    await message.answer("Введіть дату Вашого заняття\n"
                         "в форматі DD.MM.25",
reply_markup=profile(current_date))


@router.message(FormAthletics.discipline)
async def incorrect_form_date(message: Message, state: FSMContext):
    await message.answer("Виберіть з наданих варіантів")


@router.message(FormAthletics.date, F.text.regexp(r"^[0-3][0-9]\.[0-1][0-
9]\.2[5-9]$"))
async def form_date(message: Message, state: FSMContext):
    await state.update_data(date=message.text)
    await state.set_state(FormAthletics.time)
    await message.answer("Введіть час початку Вашого заняття\n"
                         "в форматі ГГ:XX\n"
                         "де ГГ-години, XX-хвилини")


@router.message(FormAthletics.date)
async def incorrect_form_date(message: Message, state: FSMContext):
```

```
    await message.answer("Введіть дату в форматі DD.MM.25")


@router.message(FormAthletics.time, F.text.regexp(r"^[0-2][0-9]:[0-5][0-
9]$"))
async def form_time(message: Message, state: FSMContext):
    await state.update_data(time=message.text)
    await state.set_state(FormAthletics.distance)
    await message.answer("Введіть дистанцію в метрах\n"
                         "виключно цифрами")


@router.message(FormAthletics.time)
async def incorrect_form_time(message: Message, state: FSMContext):
    await message.answer("Введіть час в форматі ГГ:ХХ\n"
                         "де ГГ-години, ХХ-хвилини")


@router.message(FormAthletics.distance, F.text.regexp(r"^[0-9]+$"))
async def form_distance(message: Message, state: FSMContext):
    await state.update_data(distance=message.text)
    await state.set_state(FormAthletics.result)
    await message.answer("Введіть Ваш результат подолання дистанції\n"
                         "в форматі  ГГ:ХХ:СС\n"
                         "де ГГ-години, ХХ-хвилини, СС-секунди")


@router.message(FormAthletics.distance)
async def incorrect_form_distance(message: Message, state: FSMContext):
    await message.answer("Введіть дистанцію виключно цифрами")


@router.message(FormAthletics.result, F.text.regexp(r"^[0-2][0-9]:[0-5][0-
9]:[0-5][0-9]$"))
async def form_result(message: Message, state: FSMContext):
    await state.update_data(result=message.text)
    await state.set_state(FormAthletics.photo)
    await message.answer("Завантажте фото вашого тренування\n")


@router.message(FormAthletics.photo, F.photo)
async def form_photo(message: Message, state: FSMContext):
    await get_photo(message, bot)
    await message.answer("Дані записано!\n"
                         "Гарного дня. Бережіть себе!")

    data = await state.get_data()
    date_linux = data.get("date")
    await add_base_athl(data.get("discipline"), date_linux, data.get("time"),
                        data.get("distance"), data.get("result"),
message.from_user.id)
    await state.clear()

    surname, name =  await get_base_surname_name(message.from_user.id)
    result_text = (f'Результат записано!\n'
                   f'прізвище та ім\'я: {surname} {name}\n'
                   f'вид спорту: {data.get("discipline")}\n'
                   f'дата тренування: {data.get("date")}\n'
                   f'початок тренування: {data.get("time")}\n'
                   f'дистанція: {data.get("distance")}\n'
                   f'результат: {data.get("result")}\n'
                   )

    await bot.send_message(ID_ADMIN, text=result_text)
```

```
    await bot.send_message(ID_SUPERADMIN, text=result_text)
    await message.answer("Бажаєте завантажити фото/відео \n"
                         "з Вашого тренування", reply_markup=video_photo_kb)


@router.message(FormAthletics.photo)
async def incorrect_form_date(message: Message, state: FSMContext):
    await message.answer("Ви не надіслали фото")


@router.message(FormAthletics.result)
async def incorrect_form_result(message: Message, state: FSMContext):
    await message.answer("Введіть час в форматі XX:XX:XX")
```

handlers/form_registration.py

```
from aiogram import Router, F
from aiogram.types import Message
from aiogram.fsm.context import FSMContext
from utils.states import FormRegistration
from utils.db_utils import add_base_empl
from keyboards.builders import profile
from main import bot, ID_ADMIN, ID_SUPERADMIN

router = Router()


@router.message(F.text.lower() == 'реєстрація')
async def fill_profile(message: Message, state: FSMContext):
    await state.set_state(FormRegistration.surname)
    await message.answer("Доброго дня! \nДавайте знайомитись! \nВведіть своє
прізвище")


@router.message(FormRegistration.surname)
async def form_surname(message: Message, state: FSMContext):
    await state.update_data(surname=message.text)
    await state.set_state(FormRegistration.name)
    await message.answer("Введіть своє ім'я")


@router.message(FormRegistration.name)
async def form_name(message: Message, state: FSMContext):
    await state.update_data(name=message.text)
    await state.set_state(FormRegistration.father_name)
    await message.answer("Введіть по батькові")


@router.message(FormRegistration.father_name)
async def form_father_name(message: Message, state: FSMContext):
    await state.update_data(father_name=message.text)
    await state.set_state(FormRegistration.region)
    await message.answer("Введіть свій регіон",
                         reply_markup=profile(["Київщина", "Полтавщина",
                                               "Сумщина", "Чернігівщина",
                                               "Волинь", "Житомирщина",
                                               "Харківщина", "Тернопільщина",
                                               "Хмельниччина", "Вінниччина",
                                               "Черкащина", "Рівненщина",
                                               "Львівщина"],
                                               1, 2, 2, 2, 2, 2, 2))
```

```
@router.message(FormRegistration.region,
                    F.text.casefold().in_(["київщина",
                                           "полтавщина", "сумщина",
                                           "чернігівщина", "волинь",
                                           "житомирщина", "харківщина",
                                           "тернопільщина", "хмельниччина",
                                           "вінниччина", "черкащина",
                                           "рівненщина", "львівщина"]))


async def form_region(message: Message, state: FSMContext):
    await state.update_data(region=message.text)
    await state.set_state(FormRegistration.company)
    await message.answer("Введіть свою компанію")


@router.message(FormRegistration.region)
async def incorrect_form_date(message: Message, state: FSMContext):
    await message.answer("Виберіть регіон з наданих варіантів")


@router.message(FormRegistration.company)
async def form_company(message: Message, state: FSMContext):
    await state.update_data(company=message.text)
    await state.set_state(FormRegistration.speciality)
    await message.answer("Введіть свою спеціальність")


@router.message(FormRegistration.speciality)
async def form_speciality(message: Message, state: FSMContext):
    await state.update_data(speciality=message.text)
    await state.set_state(FormRegistration.phone)
    await message.answer("Введіть свій номер телефону")


@router.message(FormRegistration.phone)
async def form_phone(message: Message, state: FSMContext):
    await state.update_data(phone=message.text)
    await state.set_state(FormRegistration.email)
    await message.answer("Введіть свою адресу електронної пошти")


@router.message(FormRegistration.email)
async def form_email(message: Message, state: FSMContext):
    await state.update_data(email=message.text)
    await state.set_state(FormRegistration.base_sport)
    await message.answer("Вкажіть спорт яким займаєтесь")


@router.message(FormRegistration.base_sport)
async def form_base_sport(message: Message, state: FSMContext):
    await state.update_data(base_sport=message.text)
    await state.set_state(FormRegistration.sport)
    await message.answer("Вкажіть вид спорту, в якому бажаєте взяти участь як
професіонал?",
                         reply_markup=profile(["БІГ", "ВЕЛО", "ДУАТЛОН"], 3))


@router.message(FormRegistration.sport, F.text.casefold().in_(["біг", "вело",
"дуатлон"]))
async def form_sport(message: Message, state: FSMContext):
    await state.update_data(sport=message.text)
```

```
    await state.set_state(FormRegistration.category)
    await message.answer("Вкажіть спортивну категорію")


@router.message(FormRegistration.sport)
async def incorrect_form_date(message: Message, state: FSMContext):
    await message.answer("Виберіть з наданих варіантів")


@router.message(FormRegistration.category)
async def form_category(message: Message, state: FSMContext):
    await state.update_data(category=message.text)
    await message.answer("Дякуємо за Вашу зацікавленість!\n"
                         "Гарного дня. Бережіть себе!")
    data = await state.get_data()
    await add_base_empl(data.get("surname"), data.get("name"),
data.get("father_name"), data.get("region"),
                        data.get("company"), data.get("speciality"),
data.get("phone"), data.get("email"),
                        message.from_user.id, data.get("base_sport"),
data.get("sport"), data.get("category"))
    await state.clear()
    result_text = (f'Новий клієнт!\n'
                   f'ID: {message.from_user.id}\n'
                   f'прізвище: {data.get("surname")}\n'
                   f'ім\'я: {data.get("name")}\n'
                   f'по батькові: {data.get("father_name")}\n'
                   f'регіон: {data.get("region")}\n'
                   f'компанія: {data.get("company")}\n'
                   f'спеціальність {data.get("speciality")}\n'
                   f'телефон: {data.get("phone")}\n'
                   f'пошта: {data.get("email")}\n'
                   f'спорт, яким займається: {data.get("base_sport")}\n'
                   f'спорт, яким хоче займатися: {data.get("sport")}\n'
                   f'спортивна категорія: {data.get("category")}'
                   )

    await bot.send_message(ID_ADMIN, text=result_text)
    await bot.send_message(ID_SUPERADMIN, text=result_text)
```

handlers/user_commands.py

```
from aiogram import Router
from aiogram.filters import CommandStart
from aiogram.types import Message
from keyboards.builders import profile
from utils.commands import set_commands
from utils.db_utils import get_base_telegramID
from main import bot, ID_ADMIN, ID_SUPERADMIN

router = Router()


@router.message(CommandStart())
async def start(message: Message):
    await set_commands(bot)
    await bot.send_message(ID_ADMIN, text=f'Бот запущено !!!
{message.from_user.id}')
    await bot.send_message(ID_SUPERADMIN, text=f'Бот запущено !!!
{message.from_user.id}')
    testID = await get_base_telegramID(message.from_user.id)
```

```
    if message.from_user.id == ID_ADMIN or message.from_user.id ==
ID_SUPERADMIN:
        await message.answer(f"Вітаю,
<b>{message.from_user.first_name}!</b>\n"
                             f"Почніть адміністрування",
reply_markup=profile(["АДМІНІСТРУВАННЯ"], 1))
    else:
        if testID is False:
            await message.answer(f"Вітаю,
<b>{message.from_user.first_name}!</b>\n"
                                 f"Почніть реєстрацію",
reply_markup=profile(["РЕЄСТРАЦІЯ"], 1))
        else:
            await message.answer(f"Вітаю,
<b>{message.from_user.first_name}!</b>\n"
                                 f"Ви вже зареєстровані, зафіксуйте
резутьтати",
                                 reply_markup=profile(["ФІКСАЦІЯ
РЕЗУЛЬТАТІВ"], 1))
```

utils/db_utils.py

```
from sqlalchemy.orm import Session
from aiogram.types import FSInputFile, Message
from aiogram import Bot
from openpyxl import load_workbook
from utils.db import engine, MyEmployees, MyAthletics
from main import ID_ADMIN, ID_SUPERADMIN, bot


list_empl = ["surname", "name", "father_name", "region", "company",
             "speciality", "phone", "email", "telegramID", "base_sport",
             "sport", "category"]

list_all = ["surname", "name", "father_name", "region", "company",
            "speciality", "phone", "email", "telegramID", "base_sport",
            "sport", "category", "discipline", "date", "time", "distance",
"result"]

fn = 'D:\\Program\\Python\\PyTelBot_Sport_Universal_Bot\\base.xlsx'
fn_photo =
'D:\\Program\\Python\\PyTelBot_Sport_Universal_Bot\\photo_result.jpg'
wb = load_workbook(fn)
ws = wb['data']


async def linux_date(data):
    date_new = data.get("date").split(".")
    date_new[1], date_new[0] = date_new[0], date_new[1]
    return ".".join(date_new)


async def cur_date(current_date):
    date_format = str(current_date).split("-")
    date_format[0], date_format[2] = date_format[2], date_format[0][2:]
    return ".".join(date_format)


async def get_photo(message: Message, bot: Bot):
    file = await bot.get_file(message.photo[-1].file_id)
```

```
    await bot.download_file(file.file_path, fn_photo)
    doc = FSInputFile(path=fn_photo)
    await bot.send_document(ID_ADMIN, document=doc, caption='Фото
результату')
    await bot.send_document(ID_SUPERADMIN, document=doc, caption='Фото
результату')


async def get_docum(bot: Bot):
    doc = FSInputFile(path=fn)
    await bot.send_document(ID_ADMIN, document=doc, caption='Результат вашого
запиту')
    await bot.send_document(ID_SUPERADMIN, document=doc, caption='Результат
вашого запиту')


async def add_base_empl(surname, name, father_name, region,
                        company, speciality, phone,
                        email, telegramID, base_sport, sport, category):
    with Session(engine) as session:

        empl_id = len(session.query(MyEmployees.id).all()) + 1
        session.add(MyEmployees(surname=surname, name=name,
father_name=father_name, region=region, company=company,
speciality=speciality, phone=phone, email=email, telegramID=telegramID,
base_sport=base_sport, sport=sport, category=category, id=empl_id))

        session.commit()


async def add_base_athl(discipline, date, time, distance, result,
id_telegram):
    with Session(engine) as session:

        athl_id = len(session.query(MyAthletics.id).all()) + 1
        date_query = session.query(MyEmployees).filter(MyEmployees.telegramID
== str(id_telegram)).all()
        empl_id = date_query[0].id

        session.add(MyAthletics(discipline=discipline, date=date,
result=result,
                                time=time, distance=distance,
employee_id=empl_id, id=athl_id))
        session.commit()


async def get_base_telegramID(id_telegram):
    with Session(engine) as session:
        date_query = session.query(MyEmployees).filter(MyEmployees.telegramID
== str(id_telegram)).all()
        return False if date_query == [] else True


async def get_base_surname_name(id_telegram):
    with Session(engine) as session:
        date_query = session.query(MyEmployees).filter(MyEmployees.telegramID
== str(id_telegram)).all()
        surname = date_query[0].__dict__.get("surname")
        name = date_query[0].__dict__.get("name")
        return surname, name


async def get_base_query(value, message, param, value2=None):
    with Session(engine) as session:
```

```
        if param == "all":
            date_query = session.query(MyAthletics,
MyEmployees).filter(MyEmployees.id == MyAthletics.employee_id).all()

        elif param == "surname":
            date_query = session.query(MyAthletics,
MyEmployees).filter(MyEmployees.surname == value,

MyEmployees.id == MyAthletics.employee_id).all()
        elif param == "region":
            date_query = session.query(MyAthletics,
MyEmployees).filter(MyEmployees.region == value,

MyEmployees.id == MyAthletics.employee_id).all()
        elif param == "company":
            date_query = session.query(MyAthletics,
MyEmployees).filter(MyEmployees.company == value,

MyEmployees.id == MyAthletics.employee_id).all()
        elif param == "discipline":
            date_query = session.query(MyAthletics,
MyEmployees).filter(MyAthletics.discipline == value,

MyEmployees.id == MyAthletics.employee_id).all()
        elif param == "one_date":
            date_query = session.query(MyAthletics,
MyEmployees).filter(MyAthletics.date == value,

MyEmployees.id == MyAthletics.employee_id).all()
        elif param == "old_date":
            date_query = session.query(MyAthletics,
MyEmployees).filter(MyAthletics.date > value,

MyEmployees.id == MyAthletics.employee_id).all()
        else:   #elif param == "two_date":
            date_query = session.query(MyAthletics,
MyEmployees).filter(MyAthletics.date >= value,

MyAthletics.date <= value2,

MyEmployees.id == MyAthletics.employee_id).all()
        if date_query:
            ws.delete_cols(1, 100)
            for row in range(1, len(date_query) + 2):
                col = 1
                for el in range(len(list_all)):
                    cell = ws.cell(row=row, column=col)
                    cell.value = list_all[el] if row == 1 else str(
                        date_query[row - 2][1].__dict__.get(list_all[el])) if
el < 12 else str(
                        date_query[row - 2][0].__dict__.get(list_all[el]))
                    col += 1

            wb.save(fn)
            wb.close()
            await get_docum(bot)
            return True
        else:
            return False


async def get_base_query_empl(message):
    with Session(engine) as session:
```

```
        date_query = session.query(MyEmployees).all()
        ws.delete_cols(1, 100)
        for row in range(1, len(date_query) + 2):
            col = 1
            for el in range(len(list_empl)):
                cell = ws.cell(row=row, column=col)
                cell.value = list_empl[el] if row == 1 else str(
                    date_query[row - 2].__dict__.get(list_empl[el]))
                col += 1

        wb.save(fn)
        wb.close()
        await get_docum(bot)
```

utils/db.py

```
from sqlalchemy import (create_engine,
                        Date, Time,
                        Column, Integer,
                        String,
                        ForeignKey)
from sqlalchemy.orm import relationship, mapped_column
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()


class BaseModel(Base):
    __abstract__ = True
    id = Column(Integer, primary_key=True, autoincrement=True)

class MyEmployees(BaseModel):
    __tablename__ = 'employees'

    surname = Column(String, nullable=False)
    name = Column(String, nullable=False)
    father_name = Column(String, nullable=False)
    region = Column(String, nullable=False)
    company = Column(String, nullable=False)
    speciality = Column(String, nullable=False)
    phone = Column(String, nullable=False)
    email = Column(String, nullable=False)
    telegramID = Column(String, nullable=False)
    base_sport = Column(String, nullable=False)
    sport = Column(String, nullable=False)
    category = Column(String, nullable=False)

    athletics_id = relationship("MyAthletics", back_populates='my_employees')

class MyAthletics(BaseModel):
    __tablename__ = 'athletics'

    date = Column(Date, nullable=False)
    time = Column(Time, nullable=False)
    discipline = Column(String, nullable=False)
    distance = Column(Integer, nullable=False)
    result = Column(Time, nullable=False)
```

```
    my_employees = relationship("MyEmployees", back_populates='athletics_id')
    employee_id = mapped_column(ForeignKey("employees.id"))


engine =
create_engine("postgresql+psycopg2://m1guelten:mp2004@localhost/miggy_book")
Base.metadata.create_all(engine)
```

utils/states.py

```
from aiogram.fsm.state import StatesGroup, State


class FormRegistration(StatesGroup):
    surname = State()
    name = State()
    father_name = State()
    region = State()
    company = State()
    speciality = State()
    phone = State()
    email = State()
    base_sport = State()
    sport = State()
    category = State()


class FormAthletics(StatesGroup):
    discipline = State()
    date = State()
    time = State()
    distance = State()
    result = State()
    photo = State()


class FormQuery(StatesGroup):
    employees = State()
    start = State()
    surname = State()
    region = State()
    company = State()
    discipline = State()
    date = State()
    one_date = State()
    two_date_start = State()
    two_date_end = State()
    old_date = State()
```

utils/commands.py

```
from aiogram import Bot
from aiogram.types import BotCommandScopeDefault, BotCommand


async def set_commands(bot: Bot):
    commands = [
        BotCommand(
```

```
            command='start',
            description='Початок роботи'
        )
    ]

    await bot.set_my_commands(commands, BotCommandScopeDefault())
```

keyboards/inline.py

```
from aiogram.types import (
    InlineKeyboardMarkup,
    InlineKeyboardButton,
   )

video_photo_kb = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text="Додати фото/відео",
url="https://drive.google.com/drive/folders/1dUsI5ZJUxJjAR0VOWl1noVNcJWa6_Tub
?usp=sharing")
        ],
    ],
    resize_keyboard=True
)
```

keyboards/builders.py

```
from aiogram.types import ReplyKeyboardRemove

from aiogram.utils.keyboard import ReplyKeyboardBuilder

rmk = ReplyKeyboardRemove()


def profile(text: str | list, *args):
    builder = ReplyKeyboardBuilder()
    if isinstance(text, str):
        text = [text]
    [builder.button(text=txt) for txt in text]
    builder.adjust(*args)
    return builder.as_markup(
        resize_keyboard=True,
        one_time_keyboard=True,
        input_field_placeholder="Виберіть дії з меню"
    )
```