Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)
факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня бакалавра
(бакалавра, спеціаліста, магістра)

студента Калядіна Владислава Ігоровича
(ПІБ)

академічної групи 126-21-2
(шифр)

спеціальності 126 Інформаційні системи та технології
(код і назва спеціальності)

за освітньо-професійною програмою

«Інформаційні системи та технології»
(офіційна назва)

на тему Інформаційна підсистема для організації навчально-спортивної активності
(назва за наказом ректора)

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
кваліфікаційної роботи	проф. Гнатушенко В.В.			
розділів:				

Рецензент				

Нормоконтролер				

Дніпро
2025

ЗАТВЕРДЖЕНО:
завідувач кафедри
_____інформаційних технологій _____
_____та комп'ютерної інженерії_____
(повна назва)
_____ __Гнатушенко В.В.__
(підпис) (прізвище, ініціали)

«_____»_____2025_року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня ——бакалавра——
(бакалавра, спеціаліста, магістра)

студенту_____Калядіну В.І._____ **академічної групи** _____126-21-2____
(прізвище та ініціали) (шифр)
спеціальності _____126 «Інформаційні системи та технології»_____

за освітньою-професійною програмою _____
(за наявності)
_____«Інформаційні системи та технології»_____

на тему Інформаційна підсистема для організації навчально-спортивної активності

затверджену наказом ректора НТУ «Дніпровська політехніка» від 23.05.2024 р. .№ 469-с

Розділ	Зміст	Термін виконання
Розділ 1	Аналіз предметної області	
Розділ 2	Постановка задач Проектування програми Реалізація додаткових функцій	
Розділ 3	Тестування та валідація програми	

Завдання видано_____ _____Гнатушенко В.В._____
(підпис керівника) (прізвище, ініціали)

Дата видачі _____

Дата подання до екзаменаційної комісії_____
Прийнято до виконання _____ _____Калядін В.І._____
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 99 с., 30 рис., 1 дод., 22 джерела.

Об'єкт розробки: інтелектуальний асистент для персонального розвитку.

Мета кваліфікаційної роботи: створити інтелектуальну систему, яка допомагає користувачам покращувати свої навички у різних сферах, включаючи навчання, спорт та особистісний розвиток, шляхом динамічного аналізу дій та інтерактивної взаємодії.

У вступі проводиться аналіз та огляд сучасного стану проблеми, визначається мета кваліфікаційної роботи та область її застосування, обґрунтовується актуальність теми та формулюються завдання дослідження. Перший розділ містить аналіз предметної області, огляд існуючих аналогів та їхніх функціональних можливостей, виявлення недоліків і визначення унікальних особливостей запропонованого рішення. Другий розділ містить постановку задачі, встановлення вимог до програмної реалізації, вибір методів та алгоритмів, що забезпечують ефективне функціонування асистента. Розглядається проектування програми, описуються структура та алгоритми її роботи, визначаються вхідні та вихідні дані, характеристика параметрів технічних засобів, описується робота системи, включаючи взаємодію з користувачем. Також реалізації додаткових функцій, включаючи алгоритми персоналізації та адаптації вправ і завдань. Третій розділ містить тестування програми, перевірку її ефективності та аналіз отриманих результатів.

Ключові слова: ІНТЕЛЕКТУАЛЬНИЙ АСИСТЕНТ, PYTHON, POSTGRESQL, НАВЧАННЯ, СПОРТ, ОСОБИСТИЙ РОЗВИТОК, АНАЛІЗ ДАНИХ, ДИНАМІЧНЕ КОРИГУВАННЯ.

ABSTRACT

Explanatory note: 99 pages, 30 pics, 1 app., 22 sources.

Development object: intelligent assistant for personal development.

The goal of the qualification work: to create an intelligent system that helps users improve their skills in various areas, including education, sports, and personal development, through dynamic analysis of actions and interactive interaction..

The introduction analyzes and reviews the current state of the problem, determines the goal of the qualification work and its scope, justifies the relevance of the topic and formulates the research tasks. The first section contains an analysis of the subject area, a review of existing analogues and their functionality, identifies shortcomings and defines unique features of the proposed solution. The second section contains the statement of the problem, establishes requirements for software implementation, selects methods and algorithms that ensure the effective functioning of the assistant. The design of the program is considered, the structure and algorithms of its work are described, input and output data are determined, the characteristics of the parameters of the technical means are described, the operation of the system is described, including interaction with the user. Also, the implementation of additional functions, including algorithms for personalization and adaptation of exercises and tasks. The third section contains testing of the program, verification of its effectiveness and analysis of the results obtained.

Keywords: INTELLIGENT ASSISTANT, PYTHON, POSTGRESQL, LEARNING, SPORTS, PERSONAL DEVELOPMENT, DATA ANALYSIS, DYNAMIC ADJUSTMENT.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

GUI – Graphical User Interface (графічний інтерфейс користувача).

XP – Experience Points (досвід користувача).

Штучний інтелект (ШІ) – це галузь інформатики, яка займається розробкою інтелектуальних машин, здатних виконувати завдання, які зазвичай потребують людського інтелекту.

ПК – персональний комп'ютер.

ВСТУП

Сучасний світ потребує ефективних інструментів для саморозвитку та підвищення продуктивності. Автоматизовані асистенти стають невід'ємною частиною життя, допомагаючи людям планувати завдання, контролювати прогрес і адаптувати рекомендації відповідно до їхніх потреб.

Метою даної кваліфікаційної роботи є розробка автоматизованого асистента, який дозволяє користувачеві взаємодіяти з освітніми матеріалами, фізичними вправами та особистими цілями. У розділі "Навчання" користувач переглядає матеріали різної складності за предметами та проходить тест для підтвердження засвоєння матеріалу. У розділі "Спорт" система генерує випадкові завдання з можливістю оцінки рівня складності для адаптації під користувача. У розділі "Персональний розвиток" користувач бачить стан свого розвитку і по ньому може записувати свої цілі, відзначати їх як виконані або видаляти. Система зберігає дані та забезпечує зручний механізм їх використання.

Актуальність розробки обумовлена необхідністю створення автоматизованої системи, яка не тільки зберігає та відображає інформацію про досягнення користувача, а й адаптується до його прогресу. Розробка в цьому напрямку може застосовуватися не лише у сфері навчання, а й у професійній діяльності. Наприклад, система може допомагати працівникам у відстеженні виконаних завдань, автоматично аналізувати продуктивність та пропонувати стратегії покращення ефективності. У корпоративному середовищі вона може сприяти управлінню цілями співробітників, допомагаючи їм структурувати завдання та оптимізувати робочий процес.

Крім того, індивідуалізація підходу до користувача відкриває нові можливості для створення персоналізованих навчальних траєкторій та програм фізичної активності. Завдяки збору та аналізу даних асистент може не лише підлаштовувати рівень складності вправ і завдань, а й спрогнозувати подальший розвиток користувача на основі його попередніх результатів. Це дозволяє максимально ефективно організувати процес самонавчання та

саморозвитку, що є надзвичайно важливим у сучасному динамічному суспільстві.

На основі зібраних даних асистент допомагає користувачеві оптимізувати навчальний процес, контролювати фізичні вправи, відстежувати особистісний ріст та підвищувати продуктивність у робочій сфері. Персоналізовані рекомендації та можливість гнучкої адаптації завдань сприяють підвищенню мотивації користувачів до досягнення поставлених цілей.

Об'єкт розробки – автоматизований асистент для персонального розвитку. Предмети дослідження для розробки – алгоритми аналізу даних користувача, методи персоналізації рекомендацій та інтеграція автоматизованих функцій у програмну систему.

Основними завданнями даної роботи є:

- аналіз існуючих рішень у сфері автоматизованих асистентів;
- проектування та реалізація системи, що дозволяє користувачам взаємодіяти із завданнями навчання та спорту;
- розробка алгоритмів персоналізації для адаптації вправ;
- реалізація механізму збереження даних користувача та їх подальшого аналізу;
- тестування та оцінка ефективності роботи системи.

У процесі розробки використовувалися такі технології: Python для основної логіки, інструмент pgAdmin 4 для управління та адміністрування баз даних PostgreSQL, а також алгоритми аналізу інформації для динамічної адаптації завдань.

Отримані результати можуть бути використані для подальшого розвитку персональних, навчальних та фізичних програмних засобів. Впровадження таких рішень сприятиме підвищенню якості особистісного розвитку, розширенню можливостей самоорганізації та професійного зростання користувачів.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Загальні відомості

У сучасному світі автоматизовані асистенти набувають дедалі більшої популярності завдяки їх здатності спрощувати процеси саморозвитку, навчання та фізичної активності. Вони надають користувачам змогу ефективно планувати свої завдання, контролювати прогрес і адаптувати діяльність відповідно до власних потреб. Автоматизовані асистенти часто застосовуються в різних сферах, таких як освіта, медицина, менеджмент, бізнес, спорт та особистісний розвиток.

Особливо важливим напрямом у таких системах є відстеження цілей та досягнень користувача – ключовий інструмент, що дозволяє не лише бачити результативність виконаних завдань, а й формувати мотивацію на майбутнє. Систематичний облік досягнень сприяє підвищенню самооцінки, дисципліни та ефективного планування часу, незалежно від сфери застосування: навчання, кар’єра, спорт чи особистісні виклики.

Автоматизований асистент, розроблений у межах даної роботи, орієнтований на підтримку користувача в навчанні, фізичних тренуваннях та особистісному розвитку. Його основною функцією є надання зручного інтерфейсу для роботи з навчальними матеріалами, генерації завдань для фізичної активності, відстеження персональних цілей та досягнень, а також ведення особистих нотаток. З додаткових функцій використовується міні-чіт-бот для комунікації з користувачем.

На відміну від традиційних додатків, даний асистент не просто фіксує цілі, а й пропонує динамічну взаємодію. Наприклад, у фізичних вправах користувач може обирати в якому напрямку він хоче виконувати вправи, після чого оцінює складність виконання. Це дозволяє системі адаптувати навантаження та формувати індивідуальний план розвитку. У навчальних цілях реалізовано матеріали різного рівня які відкриваються після засвоєння

попередніх матеріалів за допомогою тестів, а також в персональному розвитку показано діаграму його даних по якому може створювати та зберігати свої персональні цілі. Завдяки таким рішенням асистент сприяє не лише плануванню, а й реальному прогресу в саморозвитку.

У контексті цифровізації освітнього процесу та особистісного зростання, важливо забезпечити адаптивність і зрозумілість таких асистентів. Завдяки використанню сучасних фреймворків, таких як Flet, створення інтерактивного інтерфейсу стає простішим, що забезпечує ефективну взаємодію користувача з програмою та формує позитивний користувацький досвід.

1.2. Аналіз існуючих рішень

На сьогодні існує велика кількість рішень, спрямованих на навчання, спорт і саморозвиток. Серед них можна виділити:

- Системи управління навчанням (LMS). Moodle, Google Classroom, Coursera, які дозволяють проходити курси та тестування;
- Фітнес-додатки. MyFitnessPal, Nike Training Club, які генерують вправи та відстежують фізичну активність;
- Додатки для постановки цілей. Todoist, Trello, Notion, які допомагають організовувати особисті завдання та контролювати їх виконання;
- Мотиваційні трекери. Habitica, Goal Tracker, які поєднують ігрові механіки з досягненням цілей.

Незважаючи на різноманіття існуючих платформ, більшість із них зосереджені лише на одній сфері – навчанні, спорті або плануванні.

Більше того, відстеження досягнень зазвичай реалізовано поверхнево, без врахування індивідуальних особливостей користувача або можливості адаптації системи під різні типи цілей – освітні, особисті, короткострокові чи довгострокові.

Таблиця 1.1 – Порівняння функціональних можливостей популярних рішень і розроблюваного асистента.

№	Назва системи	Сфера	Відстеження прогресу	Персоналізація	Мультифункціональність	Зворотний зв'язок
1.	Coursera	Освіта	+	-	-	-
2.	MyFitnessPal	Спорт	+	+	-	-
3.	Habitica	Самороз виток	+	-	-	-
4.	Notion	Плануван ня	-	+	+	-
5.	Розроблюван а система	Усі сфери	+	+	+	+

Як видно з Таблиця 1.1, перевагою запропонованого рішення є об'єднання кількох напрямів розвитку в єдиній платформі з гнучким трекінгом прогресу та персоналізацією під потреби користувача.

Крім того, більшість додатків не враховують зворотний зв'язок від користувача щодо складності виконаних завдань чи рівня засвоєного матеріалу, що обмежує можливість подальшої адаптації системи. Запропонована система вирізняється тим, що включає механізм самооцінки після кожної активності, що дозволяє алгоритмам (у майбутньому – на основі нейромереж) коригувати складність і підбирати більш відповідні завдання. Це відкриває шлях до персоналізованого навчання та тренувань.

Унікальність розроблюваного застосунку полягає в інтеграції всіх ключових напрямів у єдину систему, з фокусом на гнучкому та візуально

зручному трекінгу цілей та досягнень, що дозволяє користувачеві комплексно підходити до саморозвитку.

1.3. Основні проблеми та виклики

Розробка програмного застосунку для відстеження цілей та досягнень користувача потребує вирішення кількох ключових проблем:

- Персоналізація. Адаптація контенту, завдань і відображення прогресу відповідно до поведінки та уподобань користувача;
- Мотиваційні механізми. Впровадження систем винагород або візуалізацій, які стимулюють досягнення цілей;
- Збереження та аналіз даних. Ефективне управління прогресом без втрати інформації, можливість перегляду історії досягнень;
- Зручний інтерфейс. Забезпечення простої та зрозумілої взаємодії з додатком, включаючи візуальне представлення прогресу;
- Гнучкість та масштабованість. Можливість додавання нових типів цілей, категорій активностей та інтеграції з іншими сервісами (наприклад, календарями, нагадуваннями або платформами навчання).

Ще одним важливим аспектом є реалізація механізмів зворотного зв’язку після виконання завдань. Зокрема, у фізичних вправах користувачі можуть оцінювати складність, що дозволяє системі автоматично регулювати навантаження. У навчальних активностях передбачена можливість фіксації тем, які були засвоєні, що позначається на інтерфейсі (наприклад, зміною кольору кнопки) і дозволяє швидко орієнтуватися в прогресі. Це створює інтерактивний і гнучкий підхід до розвитку.

Складність також полягає в реалізації універсальної структури цілей, яка б дозволяла користувачам створювати власні категорії, визначати пріоритетність і контролювати досягнення у довільний спосіб – як через текстові описи, так і через графічні індикатори чи шкали прогресу.

1.4. Психологічні аспекти навчально-спортивного саморозвитку та роль цифрових інструментів

Саморозвиток у сфері навчально-спортивної активності – це не лише набуття нових знань чи досягнення фізичних результатів. Це також комплексний психологічний процес, який включає самоусвідомлення, постановку цілей, формування внутрішньої дисципліни та розвиток емоційної стійкості.

Дослідження вказують, що наочне відстеження прогресу в навчанні та тренуваннях сприяє зростанню мотивації, формує позитивні звички та підвищує рівень задоволення від процесу. Візуалізація досягнень, поступове ускладнення завдань і підтримка цифрових інструментів сприяють зміцненню психологічної стабільності та покращенню емоційного стану.

Інформаційні підсистеми, які надають персоналізовані рекомендації та забезпечують зворотний зв’язок, дозволяють:

- формувати позитивну динаміку саморозвитку;
- знижувати тривожність завдяки структурованості та зрозумілим цілям;
- стимулювати внутрішню мотивацію, яка є стійкішою за зовнішню;
- адаптувати навантаження до індивідуального психофізіологічного стану користувача.

Таким чином, цифрові інструменти виступають у ролі електронного тренера та наставника, що супроводжує користувача у процесі досягнення освітніх і фізичних результатів.

1.5. Потреби цільової аудиторії

Перед початком проектування інформаційної підсистеми необхідно визначити її цільову аудиторію. Розуміння потреб користувачів дозволяє формувати доцільний функціонал, зручний інтерфейс та логічну структуру взаємодії. У межах дипломної роботи в Таблиця 1.2 виділено кілька основних категорій користувачів.

Таблиця 1.2 – Порівняння функціональних можливостей популярних рішень і розроблюваного асистента.

№	Категорія користувачів	Основні потреби
1.	Учні та студенти	Планування навчання, трекінг прогресу, підготовка до контрольних робіт і ЗНО, баланс з фізичною активністю
2.	Особи, що займаються спортом	Відстеження тренувань, адаптація фізичних навантажень, мотивація, формування звичок
3.	Молоді спеціалісти	Планування особистого розвитку, поєднання ментальної та фізичної активності, збереження балансу
4.	Люди, орієнтовані на саморозвиток	Гнучке управління особистими цілями у навчанні, спорті, розвитку навичок та звичок

Загальні потреби для всіх категорій включають:

- бачити динаміку розвитку (як у навчанні, так і у спорті);
- бачити заплановані завдання та матеріали;
- мати зручний і зрозумілий інтерфейс;
- використовувати підсистему на різних пристроях (ПК, смартфон).

Ці потреби формують вимоги до адаптивності інтерфейсу, збереження даних у зручному форматі (наприклад, PostgreSQL), а також до створення функціональної системи, яка підтримує інтеграцію навчального й спортивного модулів.

1.6. Універсальність підходу: потенціал для інших сфер

Розроблена система організації та відстеження дій, попри первинне спрямування на саморозвиток, має універсальний характер і може бути адаптована для широкого спектра сфер діяльності. Основна ідея полягає в тому, що чітка структура дій, можливість визначити виконане завдання та поступове нарощування складності ефективно працює не лише в контексті особистого зростання, а й у багатьох інших напрямках.

Освітні платформи. Система може використовуватись як основа для створення адаптивних навчальних платформ, де:

- Кожна тема має декілька тестів;
- Після проходження текстів користувачу відкривається більш складні матеріали до теми;
- Зберігається прогрес.

Фітнес та здоров’я. У сфері фізичного розвитку система дозволяє:

- Відслідковувати виконання вправ і навантаження;
- Реєструвати рівень складності після кожного тренування (легко, нормально, складно);
- Автоматично адаптувати наступне завдання відповідно до зворотного зв'язку користувача.

Таким чином, можна створити персоналізований фітнес-помічник, що підлаштовується під користувача в реальному часі.

Психологічна самодопомога та коучинг. У сфері ментального здоров’я подібна система здатна:

- Пропонувати завдання (наприклад: дихальні вправи);
- Відмічати їх виконання;
- Вести облік настрою або емоційного стану;
- Надавати рекомендації на основі попередніх дій.

Це створює структуровану систему самопідтримки або навіть основу для коучингових програм. Бізнес і управління проектами. Завдяки гнучкій структурі підходу, систему можна використати у:

- Внутрішніх корпоративних інструментах для підвищення продуктивності команд;
- Управлінні завданнями з можливістю відмітки прогресу;
- Побудові моделей відстеження ефективності працівників або команд, заснованих на щоденній активності.

1.7. Огляд наукових джерел та публікацій

У процесі розробки інформаційної підсистеми для організації навчально-спортивної активності було проаналізовано низку наукових джерел, що стосуються гейміфікації, цифрових помічників, навчання та особистісного розвитку. Основна увага приділялася публікаціям, які досліджують вплив гейміфікованих механізмів на мотивацію користувача, а також ефективність інтерактивних інструментів у формуванні звичок.

Дослідження Sailer et al. (2017) у журналі Computers in Human Behavior демонструють, що гейміфікація суттєво покращує залучення користувачів у навчальних середовищах, особливо коли використовуються такі елементи, як бали, рівні, досягнення, значки та рейтинг. Ці елементи стимулюють прогрес, дають зворотний зв’язок і створюють емоційне залучення до процесу.[19]

Deterding et al. (2011) у своєму огляді From Game Design Elements to Gamefulness розкривають концепцію гейміфікації як перетворення повсякденних завдань у «ігровий досвід», що базується на внутрішній мотивації – прагненні до автономії, розвитку навичок та досягнень.[20]

Дослідження Gartner (2023) висвітлює прогноз щодо зростання популярності платформ персонального розвитку з елементами ШІ, візуалізації прогресу та адаптивності. Вказується, що такі інструменти зможуть краще утримувати увагу користувачів і підвищувати результативність самонавчання.[21]

Окрему увагу варто приділити роботі Чарлза Дахигга The Power of Habit, де розглядається нейропсихологічний механізм формування звички: сигнал – дія – нагорода. Цей підхід безпосередньо застосовано у структурі модуля персональних цілей, де користувач виконує дію, отримує візуальне підтвердження та зберігає мотивацію завдяки підвищенню рівня/досвіду.[22]

Таблиця 1.3 – Огляд використаних наукових джерел.

№	Джерело	Автор	Основна ідея
1.	Gamification in Education: What, How, Why Bother? (Computers in Human Behavior, 2017)	Sailer, Hense, Mayr, Mandl	Вплив гейміфікації на мотивацію користувачів у навчальних середовищах
2.	From Game Design Elements to Gamefulness (2011)	Deterding, Dixon, Khaled, Nacke	Гейміфікація як підхід до підвищення залученості та мотивації
3.	Gartner Research 2023: Digital Self-Improvement Platforms	Gartner Group	Прогноз розвитку платформ особистісного росту на основі ШІ

1.8. Зарубіжний досвід

На міжнародному рівні вже реалізовано низку успішних цифрових платформ, які спрямовані на розвиток користувача в контексті навчання, спорту та особистісного зростання. Аналіз цих рішень дозволяє не лише зрозуміти їхню ефективність, а й адаптувати найкращі практики до вітчизняних умов.

Strava – одна з найпопулярніших світових платформ для відстеження спортивної активності. Серед основних функцій – GPS-трекінг, аналіз

результатів тренувань, соціальні взаємодії (лайки, коментарі), а також «челенджі», які мотивують до участі у змаганнях. Strava дозволяє створювати звички через регулярну активність і візуалізацію прогресу (графіки, сегменти). Цей підхід тісно пов'язаний із гейміфікацією, де користувач змагається як із собою, так і з іншими.

Khan Academy – безкоштовна освітня платформа, що надає доступ до відеоуроків, інтерактивних вправ та персоналізованих навчальних маршрутів. У системі реалізовано нагороди за прогрес, рейтинги та графіки успішності. Користувач отримує «енергію», «баджи» та звіти про успіхи. Основна її сила – адаптивність, що дозволяє учням працювати у власному темпі.

Duolingo – популярний застосунок для вивчення іноземних мов, який демонструє високий рівень гейміфікації. Користувачі отримують очки, досягають нових рівнів, відкривають «скарбнички» та конкурують у турнірних таблицях. Система пропонує щоденні цілі, streak-індикатори, віртуальну валюту, що утримує мотивацію.

Таблиця 1.3 – Порівняльна таблиця.

№	Платформа	Основна мета	Гейміфікація	Адаптивність	Охоплення сфер	Користувачів
1.	Strava	Спорт	Так (челенджі, лайки)	Часткова	Тільки спорт	100+ млн
2.	Khan Academy	Освіта	Так (бали, бейджі)	Так	Тільки навчання	135+ млн
3.	Duolingo	Вивчення мов	Так (рівні, streak)	Так	Лінгвістика	500+ млн
4.	Запропонована система	Навчання + спорт + цілі	Так (оцінка дій, прогрес, рівні)	Так	Комплексно	Персонально

1.9. Роль штучного інтелекту в цифрових ассистентах

Штучний інтелект (ШІ) поступово стає важливою складовою сучасних інформаційних систем. У контексті цифрових асистентів ШІ може бути використаний для:

- персоналізованого підбору завдань;
- прогнозування прогресу користувача;
- адаптації інтерфейсу відповідно до поведінки;
- генерації відповідей у чат-ботах на основі природної мови;
- виявлення рівня зацікавленості та навантаження користувача в реальному часі;
- формування індивідуальних маршрутів розвитку.

У провідних IT-компаніях світу вже активно використовуються системи, які аналізують дії користувача та в режимі реального часу пропонують оптимізовані навчальні або оздоровчі стратегії. Наприклад, освітні платформи, як-от Coursera або Udemy, використовують алгоритми ШІ для пропозиції курсів, а Apple Fitness+ – для адаптації тренувань.

Попри те, що на даному етапі ШІ безпосередньо не інтегровано у розроблену систему, її архітектура дозволяє легко реалізувати подібні функції в майбутньому. Наприклад, можна використовувати моделі машинного навчання для аналізу темпів навчання або ефективності фізичних вправ, а також підказувати найбільш релевантні цілі. У перспективі можна реалізувати й такі можливості:

- впровадження мовної моделі (наприклад, на базі GPT або BERT) для покращеної роботи чат-бота;
- класифікація рівня складності тем за поведінковими даними користувача;
- виявлення емоційного стану на основі відповідей або часу реакції.

Крім того, у системі вже реалізовано міні-чат-бот, який у перспективі може бути розширено за допомогою ШІ для кращої взаємодії з користувачем,

зокрема – надання індивідуальних порад, мотиваційних повідомлень або автоматичного створення плану розвитку.

Таким чином, штучний інтелект відкриває нові горизонти для майбутнього розвитку цифрових асистентів, роблячи їх більш адаптивними, інтелектуальними та ефективними в довготривалій перспективі.

1.10. Обґрунтування вибору підходу

Розроблена система базується на модульному та гнучкому підході до організації особистого прогресу користувача. Основні принципи, на яких ґрунтується вибір підходу:

- Уніфікованість – можливість об’єднати навчання, спорт та особисті цілі в єдиному середовищі;
- Простота у використанні – інтерфейс орієнтований на інтуїтивну взаємодію без зайвого ускладнення;
- Адаптивність – система враховує дії користувача для оновлення характеристик та завдань;
- Зворотний зв’язок – користувач може оцінити складність виконаного завдання чи засвоєної теми;
- Мотиваційний компонент – реалізовано базові гейміфіковані елементи для підтримки інтересу.

Цей підхід дозволяє користувачеві залишатися активним, контролювати власний прогрес та досягати результатів в особистісному розвитку. Також у процесі розробки інформаційної підсистеми для організації навчально-спортивної активності було обрано гнучкий модульний підхід, який передбачає інтеграцію кількох сфер саморозвитку в одну систему. Такий вибір зумовлений кількома ключовими причинами:

1. Комплексність. Більшість існуючих платформ (як українських, так і міжнародних) орієнтуються переважно на одну сферу – навчання, спорт або менеджмент цілей. Запропонована ж система націлена на поєднання кількох напрямів – навчання, фізичної активності та особистого планування.

Це відповідає реальним потребам сучасної людини, яка прагне гармонійного розвитку одразу у кількох вимірах.

2. Простота та зручність. Інтерфейс побудовано таким чином, щоб він був доступним навіть для користувача без технічного досвіду. Замість перевантаження функціями – акцент на інтуїтивну навігацію: кілька кліків для оновлення прогресу, позначення навчальної теми або виконаної цілі.

3. Адаптивність. Завдяки динамічному оновленню характеристик (сила, інтелект, досвід тощо) система реагує на поведінку користувача. Це створює ефект персонального супроводу, коли результати дій безпосередньо впливають на загальний прогрес. У майбутньому така адаптивність може бути посилена нейромережею, яка може прогнозувати й пропонувати завдання згідно з індивідуальним стилем користувача.

4. Зворотний зв’язок. Після кожної дії користувач має можливість оцінити складність, що дозволяє системі враховувати рівень навантаження. Цей механізм покращує персоналізацію і формує відчуття контролю з боку користувача. Такий принцип використовується в Google Forms при створенні опитувань з оцінкою складності або задоволеності – і це підвищує залученість користувачів.

5. Мотивація через гейміфікацію. Система базується на гейміфікований логіці: наявність рівнів, прогресу, індикаторів навичок. Це підтримує внутрішню мотивацію, оскільки користувач бачить результати та прагне поліпшень. Згідно з [Sailer et al., 2017], візуалізація прогресу, рівнів та досягнень підвищує утримання користувачів у системі в середньому на 30–40%.

Обраний підхід базується на реальних потребах цільової аудиторії, враховує успішні світові практики (Khan Academy, Strava, Duolingo), та дозволяє створити збалансований інструмент саморозвитку, орієнтований на зручність, ефективність і адаптивність.

1.11. Висновки до розділу

У першому розділі було проведено всебічний аналіз предметної

області, пов’язаної з організацією навчально-спортивної активності за допомогою цифрових рішень. Розглянуто поточний стан технологій, що сприяють саморозвитку користувача, виявлено недоліки існуючих рішень, а також обґрунтовано потребу у створенні універсальної системи, яка об’єднує навчання, спорт і особистісне планування в єдиному середовищі.

Визначено ключові технічні, функціональні та психологічні аспекти майбутньої системи, які формують основу її ефективного застосування. Значну увагу приділено аналізу потреб цільової аудиторії, що дозволило адаптувати функціонал під конкретні сценарії використання.

У розділі також висвітлено огляд наукових джерел та публікацій, які підтверджують доцільність застосування гейміфікації, систем мотивації та персоналізованих рекомендацій у цифрових ассистентах. Показано, що наявність візуального прогресу, адаптації навантаження та простого інтерфейсу має значний позитивний вплив на залучення користувача.

На прикладі зарубіжного досвіду (Khan Academy, Duolingo, Strava) доведено, що подібні платформи досягають високих результатів завдяки персоналізації, ігровим механікам та активному зворотному зв’язку. Запропонована ж система, на відміну від них, орієнтована не лише на одну сферу, а на комплексний підхід до розвитку, що включає ментальний, фізичний та емоційний виміри.

У підрозділі 1.10 обґрунтовано вибір підходу до реалізації системи: модульність, зручність, адаптивність та вмотивованість стали ключовими критеріями, які забезпечують довготривале використання системи без втрати зацікавленості з боку користувача.

Таким чином, перший розділ закладає ґрунтовну теоретичну базу для подальшого проектування і реалізації інформаційної підсистеми, яка здатна адаптуватися до потреб користувача, підтримувати його прогрес і надавати інструменти для досягнення цілей у різних сферах життя. Отримані висновки підтверджують актуальність теми дослідження та створюють основу для ефективної реалізації поставлених завдань у наступних розділах.

РОЗДІЛ 2
ПОСТАНОВКА ЗАДАЧ КВАЛІФІКАЦІЙНОЇ РОБОТИ

2.1. Призначення розробки та область застосування

Автоматизований асистент для саморозвитку створений для допомоги користувачам у відстеженні прогресу в навчанні, фізичному розвитку та досягненні особистих цілей, які можуть вільно додавати й редагувати відповідно до власних пріоритетів. Основне призначення застосунку – надання зручного інструменту для слідкування власного розвитку шляхом інтеграції навчальних матеріалів, фізичних вправ та динамічної системи постановки й досягнення цілей.

Програма дозволяє користувачу чітко бачити свій шлях до успіху, фіксувати досягнення на різних етапах і тим самим підтримувати мотивацію та внутрішню дисципліну. Цілі можуть бути як короткостроковими (наприклад, вивчити тему, пройти тренування), так і довгостроковими (покращити фізичну форму, розвинути певну характеристику).

При необхідності можно відновити систему яка зможе використовуватися для індивідуальних користувачів для особистого розвитку, так і для організацій, що надають освітні послуги або займаються розробкою програм з моніторингу фізичної активності й ментального здоров’я. Також можливе використання в програмах коучингу, підвищення продуктивності працівників або розробці систем самооцінювання для особистих і професійних досягнень.

2.2. Постановка завдання

Основними завданнями, які вирішує система, є:

- Надання користувачу можливості переглядати навчальні матеріали та проходити тестування для надання більш складних матеріалів;
- Генерація фізичних вправ із можливістю оцінки рівня складності, що дозволяє адаптувати навантаження до користувача;

- Створення інтерфейсу для роботи з персональними цілями – постановка, редагування, відзначення досягнення цілі, відображення результатів на діаграмі;
- Автоматичне збереження прогресу користувача до бази даних, включаючи досягнуті цілі, оцінки складності, завершені вправи тощо;
- Забезпечення зручного управління даними, простого перегляду досягнень і мотиваційного візуального супроводу (наприклад, зафарбування тем, прогрес-бари тощо).

Таким чином, система виконує не лише інформаційно-довідкову функцію, а й виступає в ролі цифрового партнера в розвитку користувача. Вона забезпечує індивідуалізований підхід до кожного користувача, дозволяючи враховувати особисті вподобання, рівень підготовки та динаміку зростання.

Крім того, система створена з урахуванням можливого подальшого розширення її функціональності: у майбутньому можуть бути реалізовані додаткові напрямки розвитку, нові категорії завдань або розумні підказки на основі поведінки користувача. Це дозволяє використовувати програму не лише в поточному вигляді, а й як базу для розвитку більш комплексного інструменту підтримки особистісного зростання.

2.3. Вимоги до програми

2.3.1. Вимоги до функціональних характеристик

Програма повинна виконувати такі функції як:

- Отримувати навчальні теми, матеріали та спортивні завдання з бази даних PostgreSQL;
- Фіксувати засвоєння навчальних тем користувачем, збереження цього стану у базі даних;
- Генерувати фізичні вправи з урахуванням рівня складності та можливістю його оцінювання користувачем (легко, нормально,

складно);

- Зберігати, створювати, редагувати, видаляти та відображати особисті цілі користувача із зазначенням статусу досягнення;
- Здійснювати збереження всіх характеристик користувача (сила, витривалість, інтелект тощо) у базі даних та оновлювати їх автоматично залежно від активностей;
- Зробити автоматичне оновлення даних щоб було видно на скільки користувач далеко розвинувся, а також наочно показують його прогрес як фізично, так і розумово.

Таблиця 2.4 – Основні функції системи.

№	Функція	Опис
1.	Отримання навчальних матеріалів.	Завантаження тем з бази даних, фіксація стану засвоєння.
2.	Генерація фізичних вправ.	Формування вправ із зазначенням складності та можливістю її оцінки.
3.	Робота з цілями.	Створення, редагування, видалення та відмітка статусу цілей.
4.	Збереження даних.	Усі дії користувача зберігаються у базі даних
5.	Персоналізація.	Адаптація складності вправ відповідно до оцінки складності користувача.
6.	Візуалізація прогресу.	Відображення прогресу у навчанні та спорті, використання кольору та індикаторів.

Для кращого розуміння логіки роботи з додатком нижче наведено Таблиця 2.5 в якій написано взаємодії користувача з системою, яка ілюструє типовий сценарій використання.

Таблиця 2.5 – Взаємодії користувача з системою.

№	Крок користувача	Дія системи
1.	Ознайомлюється з навчальним матеріалом.	Завантажується теми, матеріали та тести. Також зберігання пройдених тестів.
2.	Обирає фізичні напрямки (наприклад, кардіо, гнучкість або швидкість).	Генерує випадкові вправи з параметрами повторення які будуть адаптовуватися під користувача після його оцінювання.
3.	Завершує вправу після оцінювання.	На основі оцінки змінює складність наступного завдання та оновлює дані у базі.
4.	Створює персональну ціль.	Додає ціль до списку користувача в базі даних.
5.	Позначає ціль як виконану.	Оновлює статус у базі данних, змінює вигляд кнопки.
6.	Переглядає досягнення.	Відображає поточний рівень сили, витривалості, інтелекту та інших характеристик користувача.
7.	Виходить із програми.	Всі зміни автоматично зберігаються у базі даних PostgreSQL.

2.3.2. Вимоги до інформаційної безпеки

З огляду на те, що програмне забезпечення працює з персональними даними користувача, що входять його навчальні досягнення, фізичні характеристики, оцінки складності вправ та особисті цілі, важливо забезпечити належний рівень інформаційної безпеки.

1. Усі дані користувача зберігаються у базі даних PostgreSQL, яка працює локально на пристрої користувача. Відмова від використання зовнішніх серверів або хмарних сервісів мінімізує ризик витоку інформації та відповідає вимогам конфіденційності щодо обробки персональних даних.
2. Доступ до бази даних обмежується лише поточним користувачем, який має доступ до локального середовища запуску програми. Додатково передбачено, що структура бази даних не дозволяє вносити неконтрольовані зміни безпосередньо, — редагування можливе лише через інтерфейс програми, що знижує ризик несанкціонованих дій.
3. Для підвищення безпеки передбачено регулярне автоматичне збереження даних у базі при виконанні дій користувача, а також можливість реалізації резервного копіювання для уникнення втрати інформації.
4. У майбутньому можливе впровадження шифрування даних (наприклад, використання AES або простої реалізації, як-от шифр Цезаря), що підвищить безпеку при зберіганні особливо чутливої інформації, наприклад, про здоров’я користувача.

Таким чином, програмне забезпечення задовольняє базові вимоги захисту даних та дозволяє зберігати їх без загрози витоку чи стороннього доступу. Для зручного узагальнення основні вимоги до інформаційної безпеки наведені у таблиці 2.6 .

Таблиця 2.6 – Вимоги до інформаційної безпеки.

№	Вимога	Опис
1.	Локальне зберігання даних	Усі дані зберігаються тільки на пристрої користувача
2.	Обмеження доступу до файлів	user_data.json доступний лише поточному користувачу системи.
3.	Захист від стороннього редагування	Дані можна змінити тільки за допомогою функціоналу програми.
4.	Резервне копіювання	Автоматичне збереження копії даних при виході або зміні вмісту.
5.	Можливість шифрування	Заплановано впровадження AES або шифру Цезаря для додаткового захисту.

2.3.3. Вимоги до складу та параметрів технічних засобів

Розроблене програмне забезпечення орієнтоване на використання широким колом користувачів, тому має бути доступним на більшості сучасних операційних систем та не потребувати складного встановлення.

Операційна сумісність: програма повинна безперешкодно працювати на Windows, macOS і Linux.

Мінімальні технічні вимоги:

- Процесор з частотою 1.5 ГГц або вище;
- 2 ГБ оперативної пам’яті;
- Встановлений інтерпретатор Python версії 3.8 або новішої;

- Додатково можуть бути потрібні стандартні Python-бібліотеки (наприклад, random, os) та, у випадку графічного інтерфейсу, бібліотека Flet.

Програма не вимагає інтернет-з’єднання, що робить її зручною для автономного використання, зокрема в освітніх закладах, тренінгових центрах чи під час подорожей.

2.3.4. Вимоги інформаційної та програмної сумісності

Програма повинна залишатися максимально простою та відкритою до розширення:

- Застосовуються стандартні бібліотеки Python та бібліотеки для роботи з PostgreSQL, що забезпечує кроссплатформенную сумісність і легке перенесення.
    - Усі дані зберігаються в локальній базі даних PostgreSQL, що забезпечує: структуроване та надійне зберігання; гнучкість в обробці запитів; можливість масштабування та інтеграції з іншими сервісами; простоту оновлення, перевірки та резервного копіювання.
- Структура бази даних організована у вигляді окремих таблиць для кожної функціональної частини системи (навчання, фізичні вправи, цілі, характеристики), що дозволяє у майбутньому легко додавати нові категорії без істотних змін у коді.
- Архітектура програми підтримує модульність та розширюваність, зокрема:
    - додавання нових типів вправ та навчальних напрямів;
    - нових напрямків саморозвитку;
    - впровадження нових алгоритмів персоналізації та аналізу поведінки користувача;
    - реалізацію нових форм візуалізації прогресу.

2.4. Обґрунтування вибору технологій

Під час розробки програмного забезпечення було обрано сучасні, ефективні та кросплатформні технології, що дозволяють створити зручний, масштабований і функціональний застосунок. Обраний стек забезпечує гнучкість, безпеку, легкість у розширенні, а також підтримує структуру з використанням бази даних. У таблиці 2.7 наведено використані технології та їх призначення:

Таблиця 2.7 – Обґрунтування технологій.

№	Технологія	Призначення
1.	Python	Основна мова програмування, що використовується для реалізації логіки, генерації вправ, обробки введення та керування даними.
2.	PostgreSQL	Реляційна СУБД для зберігання даних про користувача.
3.	psycopg2	Бібліотеки для взаємодії з базою даних PostgreSQL у Python. Дають змогу здійснювати CRUD-операції та абстрагують SQL-запити.
4.	Бібліотека random	Модуль для генерації випадкових вправ, що забезпечує динамічність тренувального процесу.
5.	Flet	Сучасна бібліотека для створення графічного інтерфейсу користувача на Python.
6.	datetime	Застосовується для обліку часу, дат створення вправ або цілей та аналізу прогресу користувача.
7.	matplotlib.pyplot	Створення графіків і візуалізація прогресу користувача у вигляді діаграм.
8.	numpy	Робота з числовими масивами, обчислення статистичних показників для графіків.
9.	io	Створення буфера для збереження графіків у пам’яті замість збереження в файл.
10.	base64	Кодування зображення графіка для передачі у вигляді рядка та відображення у Flet.

2.4.1. Переваги обраного стеку

Використані технології забезпечують ефективну, масштабовану та безпечну основу для розробки застосунку, що дозволяє гнучко адаптувати його до потреб користувача. Нижче наведено основні переваги обраного стеку:

1. Інтеграція з повноцінною базою даних (PostgreSQL). Замість збереження даних у JSON, використовується надійна реляційна база даних PostgreSQL, яка дозволяє працювати з великим обсягом інформації, виконувати складні запити та забезпечувати цілісність даних. Це відкриває можливості для масштабування та розширення системи.
2. Модульна архітектура з використанням psycopg2. Завдяки використанню бібліотеки psycopg2 реалізовано прямий і гнучкий доступ до бази даних PostgreSQL. Це забезпечує повний контроль над SQL-запитами та взаємодією з базою, що особливо корисно при розробці кастомізованої логіки. Такий підхід дозволяє уникнути надлишкових абстракцій та спростити налагодження, зберігаючи код структурованим та зрозумілим.
3. Сучасний графічний інтерфейс з Flet. Flet дозволяє створювати кросплатформний, інтуїтивно зрозумілий інтерфейс без використання браузера. Програма виглядає сучасно, не вимагає встановлення веб-сервера й працює автономно як десктоп-додаток.
4. Візуалізація результатів за допомогою matplotlib. Графіки й діаграми дозволяють користувачу бачити власний прогрес, покращення навичок або фізичних характеристик. Це стимулює до подальшого розвитку та дозволяє відстежувати результати у наочному вигляді.
5. Підтримка офлайн-режиму. Усі ключові функції застосунку працюють без підключення до

Інтернету. Це робить програму зручною для використання у навчальних закладах, тренінгах, поїздках або інших місцях без доступу до мережі.

6. Простота встановлення та використання. Для запуску достатньо мати встановлений Python та PostgreSQL. Інтерфейс розроблено таким чином, щоб користувач міг легко орієнтуватися навіть без технічних знань.

7. Гнучкість та розширюваність. Програма реалізована з урахуванням можливості подальшого розвитку: додавання нових категорій активностей, навичок, алгоритмів персоналізації через нейромережі або покращення візуальної складової не потребують повної перебудови системи.

Завдяки такому поєднанню технологій розроблений застосунок є не лише функціональним, але й практичним у використанні для досягнення особистих цілей, формування звичок та систематичного розвитку.

2.5. Функціональне призначення системи

Розроблена інформаційна система є автоматизованим асистентом для підтримки саморозвитку користувача та для підтримання необхідних знань та фізичної активності . Основні функції:

- Навчання – надається доступ до основних навчальних матеріалів з різних предметів, з можливістю проходження тестів які після проходження додаються більш складні матеріали з тестами.
- Спорт – система генерує випадкові фізичні вправи на основі вибраних користувачем критеріїв. Після виконання вправи користувач оцінює її складність, що дозволяє адаптувати рівень навантаження в наступних тренуваннях для досягнення оптимальних результатів.

Додатковий функціонал:

- Міні-чат-бот – допомагає користувачеві, даючи відповіді на прості питання, розповідаючи жарти чи цитати, що додає елемент

інтерактивності та розваги до системи.

- Персональні цілі – користувач може створювати та виконувати цілі в різних аспектах свого розвитку, дивлячись на діаграму характеристик. Також є можливість відмічати досягнуті цілі або видаляти їх.
- Збереження та отримання даних – використовується база даних PostgreSQL для зберігання прогресу користувача (історія вправ, засвоєні теми, досягнення) та навчальних матеріалів. Це забезпечує надійне зберігання, швидкий доступ до даних і масштабованість при роботі з великими обсягами інформації.

2.6. Опис використаних технологій та мов програмування

Для реалізації інформаційної системи було обрано мову програмування Python, яка завдяки своїй простоті, гнучкості та великій кількості бібліотек чудово підходить для створення інтерактивних застосунків. Також було використано формат JSON для зберігання користувацьких даних і навчальних матеріалів, що забезпечує просту структурування та доступність інформації.

Основні використані технології:

- Python – основна мова програмування, яка забезпечує реалізацію логіки, обробку даних, генерацію інтерфейсу та взаємодію з файлами.
- Flet – сучасна бібліотека для створення графічних інтерфейсів користувача у Python, яка дозволяє легко реалізувати кросплатформені вікна з інтерактивними елементами (кнопки, чекбокси, сторінки).
- PostgreSQL – надійна система керування базами даних, що використовується для збереження інформації про користувача: цілі, навчальні теми, вправи, оцінки тощо. Забезпечує структурованість, захист даних і можливість масштабування системи.

Додаткові бібліотеки:

- matplotlib та numpy – використовуються для створення графіків і

візуалізації даних (наприклад, прогресу навчання або фізичних характеристик). Графіки автоматично генеруються, кодуються у форматі base64 та виводяться в інтерфейсі користувача.

- random – відповідає за генерацію випадкових завдань у спортивному модулі. Завдяки цій бібліотеці система формує індивідуальні вправи, що забезпечує різноманітність тренувань і дозволяє уникнути одноманітності у процесі занять.
- io та base64 – використовуються для перетворення графіків у зображення, що можуть бути відображені в інтерфейсі без збереження у фізичні файли.
- flet – сучасна бібліотека для створення графічного інтерфейсу користувача. Забезпечує швидку розробку адаптивних і привабливих інтерфейсів, що легко налаштовуються під різні типи пристроїв. Простота використання flet дозволяє розробляти GUI без потреби в HTML або CSS, зберігаючи при цьому високу якість відображення.

Такий набір технологій забезпечує гнучкість, зручність, простоту обслуговування та розширення функціоналу у майбутньому. Програма залишається легкою у використанні як для кінцевого користувача, так і для розробника, і може бути адаптована під різні напрями розвитку та потреби.

2.7. Опис структури програми та алгоритмів її функціонування

Структура програми побудована за модульним принципом, де кожен модуль відповідає за окремий функціональний блок системи. Такий підхід забезпечує легку підтримку, розширюваність та незалежну розробку окремих частин проекту.

Інформація про користувача, навчальні теми, цілі, вправи, оцінки та ролі зберігається у базі даних PostgreSQL. Для взаємодії з базою даних використовується бібліотека psycopg2, яка дозволяє виконувати SQL-запити безпосередньо з Python-коду, забезпечуючи повний контроль над обробкою даних. Основні модулі:

- main.py – головний модуль, який ініціалізує інтерфейс користувача та запускає основні сторінки додатку (навчання, спорт, особистий розвиток, вхід/реєстрація тощо). Він відповідає за навігацію між модулями та загальну організацію взаємодії;
- education_page.py – відповідає за відображення навчальних тем та предметів. Користувач може позначати засвоєні теми. Інформація про прогрес зберігається у базі даних, що дозволяє точно відслідковувати навчальні досягнення;
- sport_page.py – модуль генерує випадкові фізичні вправи на основі характеристик користувача. Після виконання вправ користувач може оцінити їхню складність (легко, нормально, важко), що впливає на майбутнє навантаження. Усі дані зберігаються у базі;
- personal_development.py – модуль для постановки особистих цілей. Користувач може додавати, видаляти та позначати досягнуті цілі. Всі цілі зберігаються у базі даних для подальшого аналізу й відображення в інтерфейсі;
- ui_components.py – містить повторно використовувані графічні компоненти для створення інтерфейсу за допомогою бібліотеки Flet (кнопки, вікна, списки, заголовки тощо);
- logic.py – модуль для чат-бота, який відповідає на прості питання користувача, розповідає жарти або цитати, а також дає відповіді на певні команди;
- user_details_page.py – модуль для відображення та редагування характеристик користувача, таких як сила, швидкість, витривалість тощо, що змінюються залежно від виконуваних завдань.
- db.py – модуль для взаємодії з базою даних PostgreSQL. Містить SQL-запити та функції для роботи з таблицями: користувачі, навчальні теми, фізичні вправи, цілі, оцінки складності тощо. Також реалізовано підключення до бази даних, виконання транзакцій та обробку результатів за допомогою бібліотеки psycopg2.

- auth_page.py – модуль авторизації та реєстрації користувачів. Забезпечує введення логіну і пароля, перевірку облікових даних, створення нових користувачів у базі даних та перехід до основного інтерфейсу після успішного входу. Використовує db.py для перевірки користувачів.
- admin.py – модуль для адміністративного доступу. Дозволяє переглядати список користувачів. Призначений для викладачів або розробників для аналітики та тестування. Інформація виводиться з бази даних у зручному вигляді, з можливістю фільтрації та перегляду статистики.

Алгоритм роботи навчального модуля:

1. Завантаження списку предметів з бази даних PostgreSQL.
2. Відображення тем для навчання користувачу.
3. Користувач ставить позначку біля теми, яку він засвоїв.
4. Статус теми оновлюється безпосередньо в базі даних.

Алгоритм роботи спортивного модуля:

1. Користувач вибирає категорію фізичних вправ.
2. Програма випадковим чином генерує вправу з бази даних.
3. Користувач виконує вправу та оцінює її складність.
4. Система зберігає оцінку в базі даних і адаптує подальші вправи відповідно до складності, враховуючи історію оцінок.

Алгоритм роботи модуля персонального розвитку:

1. Користувач вводить новую мету.
2. Програма зберігає мету в базу даних PostgreSQL.
3. Користувач може визначити мету як виконану або видалити її, що також оновлюється у базі даних.

2.8. Опис структури даних користувача

Для збереження всіх даних про користувача використовується

реляційна база даних PostgreSQL. Взаємодія з базою здійснюється через бібліотеку psycopg2, що забезпечує пряме виконання SQL-запитів.

Основні таблиці та їх опис:

- roles — містить ролі користувачів (наприклад, user, admin).
- users — основна таблиця з інформацією про користувача: name, password_hash, age — базова інформація; strength, intelligence, speed, endurance, balance — характеристики користувача; level, experience — рівень і досвід; role_id — посилання на роль користувача; created_at, updated_at — дати створення та оновлення.
- exercise_categories — категорії фізичних вправ, які пов'язані з певними характеристиками користувача (related_stat).
- exercise_templates — шаблони вправ, що містять назву, категорію та тривалість.
- exercises — виконані або заплановані вправи користувача: user_id — власник вправи; exercise_template_id — шаблон вправи; data — JSONB з додатковими параметрами (кількість повторень, час тощо); status — поточний стан (current, completed, skipped).
- user_progress — окрема таблиця для відстеження кількості повторів вправ у кожній категорії.
- subjects — список навчальних предметів.
- topics — теми в межах кожного предмета, можуть мати попередні (обов'язкові) теми (prerequisite_topic_id).
- materials — навчальні матеріали для тем: content — текстовий зміст; difficulty_level — складність матеріалу (початковий, середній, поглиблений).
- education — індивідуальні навчальні дані користувача: subject, data (у форматі JSONB); status — статус опрацювання теми.
- user_education_progress — прогрес користувача щодо тем: test_passed — чи складено тест; materials_unlocked — чи розблоковано

матеріали; progress_counted — чи зараховано прогрес; quiz_block — блокування наступного тесту.

- quizzes, questions, answers — структура тестів: питання з рівнем складності; правильні відповіді (is_correct); результати користувача збережені в таблиці quiz_results.
- personal_goals — цілі, встановлені користувачем: goal — опис цілі; is_completed — статус виконання; updated_at — дата оновлення.

2.9. Опис роботи розробленої системи

2.9.1. Використані технічні та програмні засоби

- Операційна система: Windows ;
- Мова програмування: Python 3.10+;
- Бібліотеки: Flet, psycopg2 (для роботи з PostgreSQL), json (для обробки частини даних у форматі JSONB);
- Розробницьке середовище: PyCharm Community Edition 2022.3.2;
- Формат збереження даних: JSON;
- Інтерфейсна бібліотека: Flet;
- Формат запуску: локально на комп’ютері користувача.

2.9.2. Виклик та завантаження програми

Запустити програму можно через середовище розробки (в цій роботі запускаємо через PyCharm):

- Відкрити проєкт у PyCharm;
- Відкрити файл main.py;
- Натиснути кнопку запуску (Run).

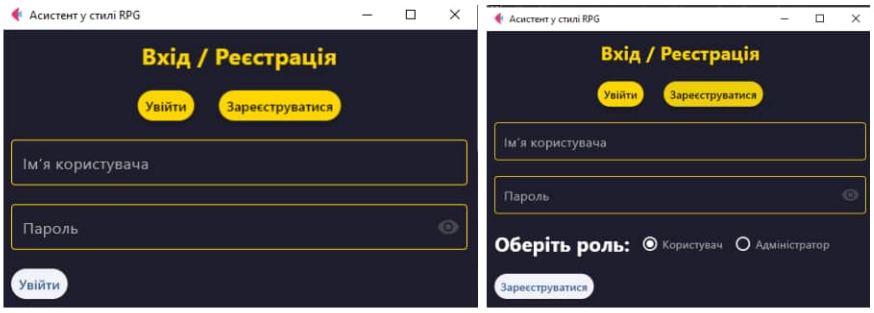
Або в командному рядку вести python main.py

2.9.3. Опис інтерфейсу користувача

Інтерфейс розробленої програми реалізований за допомогою бібліотеки Flet, що дозволяє створювати інтерактивні та кросплатформні графічні інтерфейси на мові Python. Інформація для відображення отримується з реляційної бази даних PostgreSQL, зокрема за допомогою SQL-запитів через

бібліотеку psycopg2. Користувацький інтерфейс поділяється на п’ять основних вікон, які забезпечують логічно структуровану взаємодію з основними функціями системи.

Вікно входу та реєстрації. Після запуску програми користувач потрапляє на вікно входу. Якщо в системі вже створено обліковий запис, користувач вводить логін та пароль для авторизації. Якщо ж користувач новий, йому доступна функція реєстрації, де необхідно ввести логін і пароль та обрати бути користувачем чи адміністратором. Після успішної реєстрації чи входу система автоматично показує кнопку увійти яка перенаправляє користувача до головного меню.



а) б)

Рисунок 2.0 – Вікно: а) входу; б) реєстрації.

Головна сторінка адміна. Це вікно для адміністраторів в якому відображається список всіх користувачів, яких можно видаляти та сортувати їх за ім'ям та рівнем.

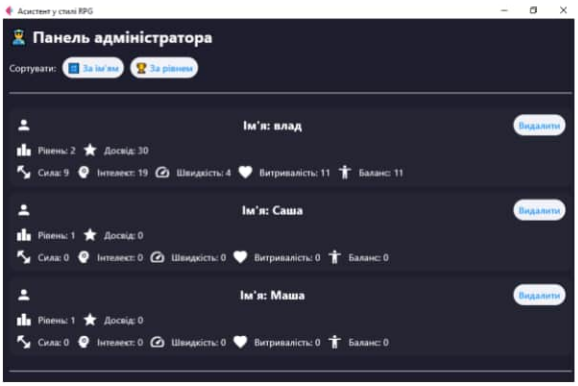


Рисунок 2.1 – Основне вікно адміна.

Головне меню користувача. Це стартове вікно програми, яке надає доступ до чотирьох основних розділів: навчання; спорт; персональні цілі; деталі користувача.

Також у цьому вікні виводяться поточний рівень і досвід користувача. Інтерфейс головного меню інтуїтивно зрозумілий і дозволяє швидко орієнтуватися між розділами програми.

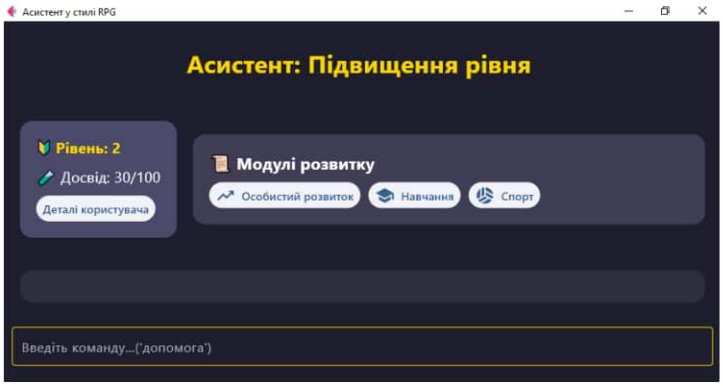


Рисунок 2.2 – Основне вікно програми.

Сторінка навчання. На цьому екрані відображається список предметів, які зберігаються у базі даних. Кожен предмет представлений у вигляді окремої кнопки: Математика; Біологія; Історія тощо.

Після вибору предмета відкривається список тем, що належать до цього предмета. Користувач має можливість ознайомитися з матеріалами кожної теми та пройти тест після чого буде доступно більш складні матеріали.

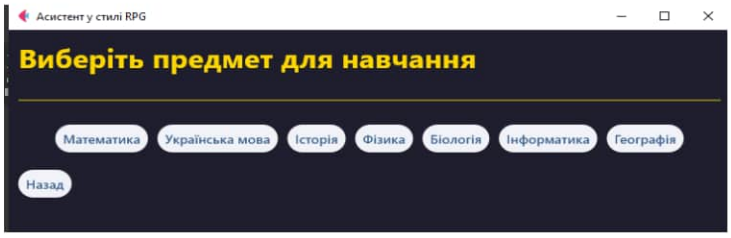


Рисунок 2.3 – Список предметів у модулі навчання.

Рисунок 2.4 – Список тем у модулі навчання.

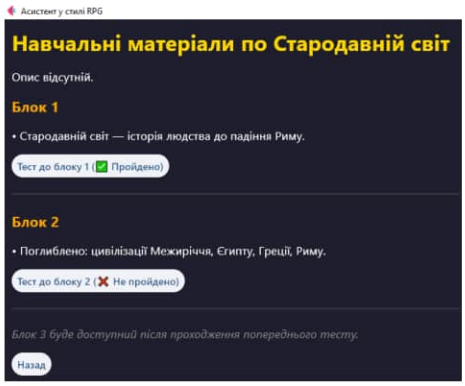


Рисунок 2.5 – Вікно з навчальними матеріалами та з тестом.

Сторінка спорту. У цьому вікні система генерує фізичну вправу, яку користувач має виконати. Після завершення вправи користувач оцінює її складність: легко; нормально; складно.

На основі цієї оцінки система адаптує подальше фізичне навантаження.

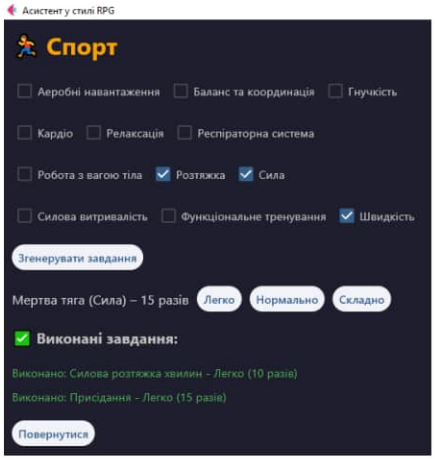


Рисунок 2.6 – Вікно з спортивними вправами.

Сторінка характеристик. Цей розділ дозволяє переглядати поточні характеристики користувача: сила; інтелект; витривалість; координація; швидкість. Характеристики змінюються автоматично залежно від активностей користувача у навчанні або спорті.

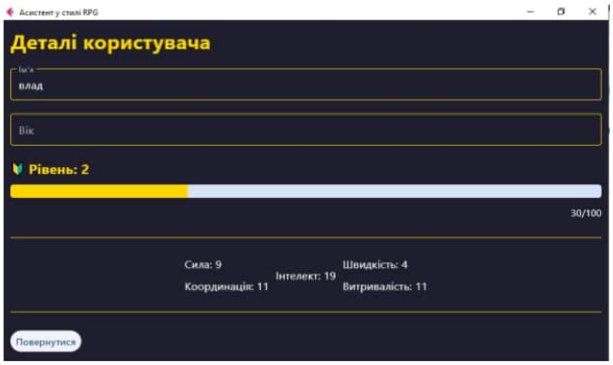
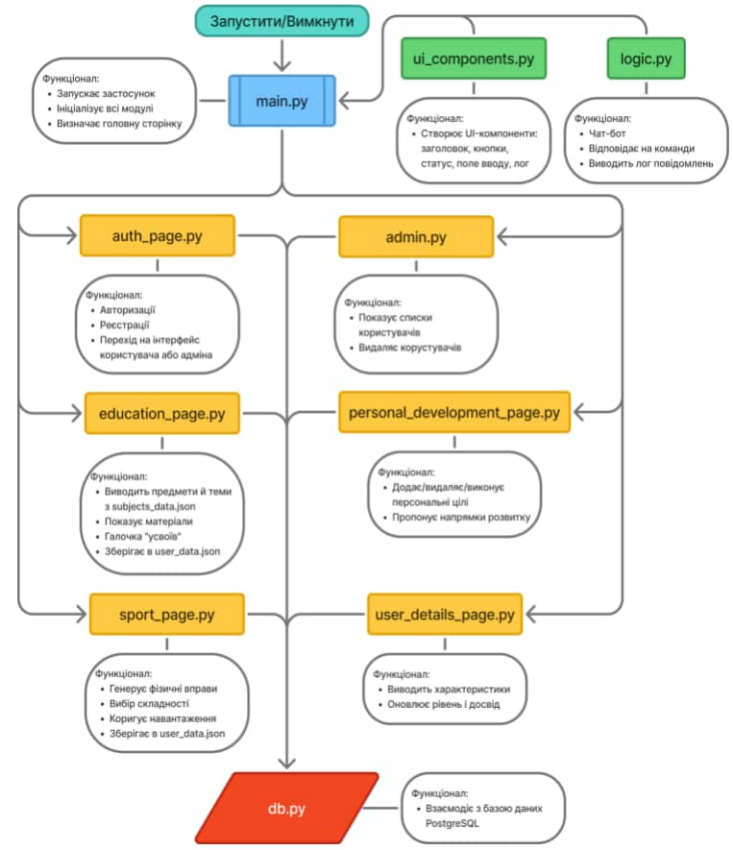
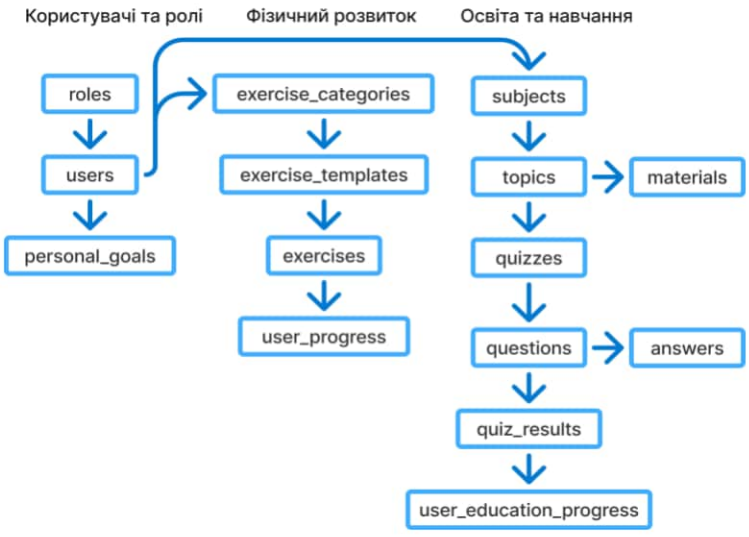


Рисунок 2.7 – Вікно з характеристиками користувача.

Нижче наведено блок-схеми які показують структуру розробленої системи, структуру базі даних та uml-діаграма яка показує взаємодію з програмою.



а)



б)

Рисунок 2.8 – Блок-схема: а) Структура розробленої системи; б) Структура бази даних

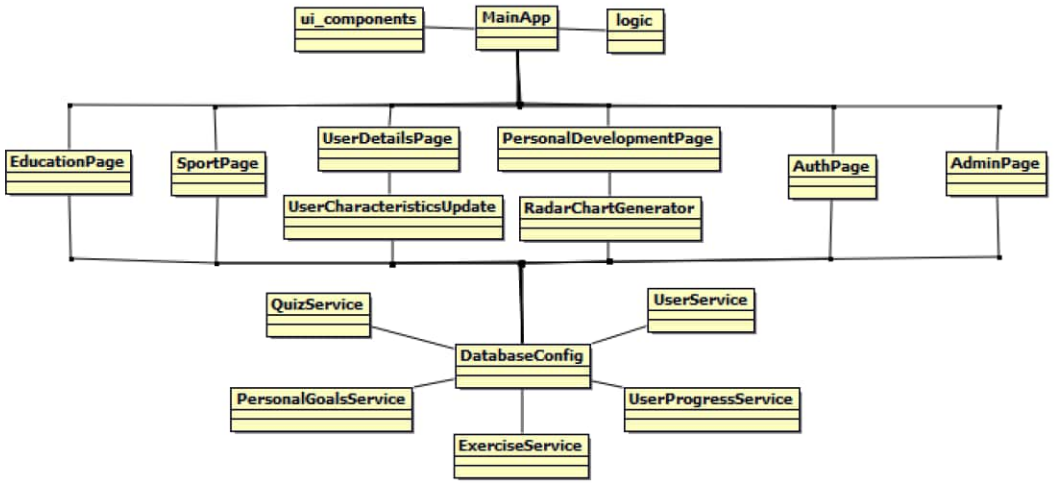


Рисунок 2.9 – Структура взаємодії з програмою.

MainApp. Файл: main.py. Опис: Це головний клас додатку який відповідає за ініціалізацію вікна, перемикання між сторінками інтерфейсу та запуск основного циклу програми.

ui_components. Файл: ui_components.py. Опис: Містить допоміжні UI-елементи, які повторно використовуються на багатьох сторінках (наприклад, кнопки, заголовки, стилі оформлення).

logic. Файл: logic.py. Опис: Модуль для чат-бота, який відповідає на прості питання користувача.

UserDetailsPage. Файл: user_details_page.py. Опис: Сторінка з характеристиками користувача такі, як сила, витривалість, інтелект тощо. Звертається до функцій з db.py для зчитування даних користувача.

UserCharacteristicsUpdater. Файл: user_details_page.py. Опис: Допоміжний клас для оновлення характеристик користувача (напр., після тренування). Використовується на сторінці UserDetailsPage.

SportPage. Файл: sport_page.py. Опис: Сторінка з фізичними вправами для користувача. Генерує завдання, які потрібно виконати. Після виконання оцінюється складність і оновлюються характеристики. Використовує DatabaseConfig (з db.py) для збереження результатів.

RadarChartGenerator. Файл: personal_development.py. Опис: Відповідає за побудову радар-графіку (павутини) для візуального відображення характеристик користувача.

PersonalDevelopmentPage. Файл: personal_development_page.py. Опис: Сторінка для ведення особистих цілей. Дає змогу користувачу вказати свої цілі, наприклад: «вивчити англійську», «підняти 30 кг». Використовує DatabaseConfig для збереження стану цілей.

EducationPage. Файл: education_page.py. Опис: Сторінка навчання. Може містити теми для вивчення, такі як математика, географія тощо. Зберігає інформацію про проходження тестів. Працює з БД через DatabaseConfig.

AuthPage. Файл: auth_page.py. Опис: Сторінка авторизації. Надає можливість входу користувача або реєстрації, зберігає/зчитує дані з бази через DatabaseConfig.

AdminPage. Файл: admin_page.py. Опис: Адміністративна панель, призначена для управління користувачами. Відображає список, та видаляє користувачів. Використовує DatabaseConfig.

DatabaseConfig. Файл: db.py. Опис: Центральний клас для роботи з базою даних. Містить функції для зчитування, оновлення, збереження інформації про користувача, прогрес, вправи тощо. До нього напряму звертаються AuthPage, UserDetailsPage, SportPage, AdminPage і т.д.

UserService. Опис: Логіка для роботи з користувачами: реєстрація, пошук, оновлення профілю.

UserProgressService. Опис: Сервіс для збереження прогресу користувача у навчанні або спорті.

ExerciseService. Опис: Сервіс для генерації/обробки вправ.

QuizService. Опис: Дані для генерації тестів у EducationPage.

PersonalGoalsService. Опис: Сервіс, що працює з особистими цілями: додавання, редагування, видалення, перевірка виконання.

2.10. Реалізація додаткових функцій

У процесі розробки системи було прийнято рішення додати низку додаткових функцій, які значно покращують взаємодію користувача з додатком, підвищують рівень персоналізації та розширюють можливості для навчання, фізичного розвитку та постановки персональних цілей. Ці функції забезпечують більш зручний і інтуїтивно зрозумілий інтерфейс, що дозволяє користувачам ефективно взаємодіяти з програмою, в той час як система адаптується до їхніх індивідуальних потреб.

У цьому розділі детально розглянуто реалізацію таких функцій, як:

- Міні-чат-бот для інтерактивної взаємодії з користувачем;
- Функція відстеження засвоєних навчальних тем;
- Система персональних цілей, що дозволяє планувати розвиток у різних напрямках;
- Інтеграція цих функцій у загальний інтерфейс програми.

2.10.1. Призначення та можливості чат-бота

Одним із доповнень до головного вікна програми став простий чат-бот, що виконує роль інтерактивного елементу для підтримки базової взаємодії з користувачем. Його головне призначення – надати короткі, миттєві відповіді на певний перелік фіксованих запитів, зробити інтерфейс більш "живим" та зручним у користуванні, такі як:

- Відповіді на поширені питання (наприклад, "Як справи?", "Що ти можеш робити?");
- Генерація випадкових жартів для розрядки атмосфери;
- Надання надихаючих цитат для мотивації користувача;
- Відображення поточного часу та дати;
- Очищення історії чату за допомогою спеціальної команди.

На відміну від повноцінних розмовних моделей, реалізований чат-бот не підтримує генерацію відкритих відповідей, а відповідає лише на ті запити, які були заздалегідь визначені у таблиці відповідностей.

Прикладами таких запитів є:

- «Як справи?»;
- «Що робити?»;
- «Жарт»;
- «Цитата»;
- «Факт»;
- «Очистити чат».

Ці запити обробляються через просту перевірку ключових слів або повних фраз, після чого користувачу надається відповідь або виводиться підготовлене повідомлення. Бот доступний лише на головному екрані програми, де він виконує роль допоміжного модуля.

Таким чином, чат-бот виконує обмежений, але корисний набір функцій, які покращують загальне враження від взаємодії з програмою, не ускладнюючи її логіку. Його реалізація дозволила створити більш привітне середовище для користувача без потреби у складних мовних моделях чи зовнішніх API.

2.10.2. Основні принципи роботи чат-бота

Міні-чат-бот працює за принципом обробки введених користувачем команд і надання відповідних відповідей згідно з наперед визначеним словником. Якщо команда невідома, бот повідомляє про це та пропонує скористатись командою “допомога”, яка відображає всі доступні опції.

Бот завжди активний і готовий відповідати на повідомлення в режимі реального часу. Це створює ефект живого діалогу й дозволяє користувачу оперативно отримувати інформацію чи просто поспілкуватися.

Таблиця 2.8 – Основні команди чат-бота.

№	Команда	Дія
1.	привіт	Вітання від бота
2.	як справи?	Бот розповідає про свій стан
3.	мені сумно	Надихає користувача
4.	хто ти?	Коротко описує себе як помічника
5.	що робити?	Відповідає, що чекає команд
6.	дякую	Висловлює вдячність
7.	до побачення	Прощається з користувачем
8.	факт	Видає цікавий факт
9.	жарт	Випадковий жарт зі списку
10.	цитата	Мотивуюча або надихаюча цитата
11.	час	Показує поточний час
12.	дата	Показує сьогоднішню дату
13.	очистити	Очищає історію чату
14.	допомога	Виводить список доступних команд

Ось приклад частини коду, яка реалізує основну логіку роботи бота:

```
# Словник основних команд
commands = {
    "факт": "Факт: Людина може жити без їжі до 3 тижнів.",
```

```
    "допомога": "Команди: факт, допомога, час, дата, жарт, цитата,
очистити.",
    "час": f"Поточний час: {datetime.now().strftime('%H:%M:%S')}",
    "дата": f"Сьогоднішня дата: {datetime.now().strftime('%d-%m-%Y')}",
    "жарт": random.choice(jokes),
    "цитата": random.choice(quotes),
    "очистити": "Лог очищено."
}
# Додаткові реакції бота на фрази користувача
mini_chat_responses = {
    "привіт": "Привіт! Як настрій? □",
    "як справи?": "Все чудово! А в тебе? □",
    "мені сумно": "Не засмучуйся! Все налагодиться. □",
    "хто ти?": "Я — твій асистент. Питай усе, що цікавить!",
    "що робиш?": "Чекаю твоїх команд. □",
    "дякую": "Завжди радий допомогти! □",
    "до побачення": "Бувай! Гарного дня! □"
}
```

Кожна команда та фраза пов’язана з відповідним текстом або дією. Наприклад, команда "жарт" викликає випадковий жарт із заздалегідь підготовленого списку.

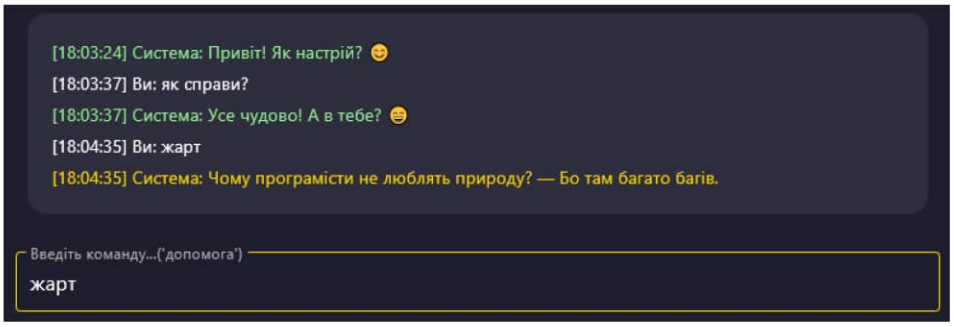


Рисунок 2.10 – Взаємодія користувача з чат-ботом.

2.11. Автоматичне оновлення характеристик користувача

2.11.1. Значення характеристик у системі

У рамках системи користувач має низку характеристик, які змінюються залежно від його активності. Це дозволяє зробити використання програми більш персоналізованим і мотивуючим.

Основні характеристики включають:

- Рівень (Level) – загальний показник розвитку користувача;
- Досвід (Experience) – накопичується в процесі виконання дій;
- Інтелект (Intelligence) – зростає при вивченні навчального матеріалу;

- Сила (Strength) – підвищується при виконанні фізичних вправ;
- Витривалість (Endurance) – також залежить від фізичної активності;
- Швидкість (Speed) – поліпшується при тренуваннях на витривалість.
- Координація (Balance) – поліпшується при тренування баланс та координації.

Ці характеристики дають змогу відслідковувати розвиток користувача в різних сферах його діяльності: фізичній, інтелектуальній та загальній.

2.11.2. Логіка оновлення характеристик

Щоразу, коли користувач виконує певну дію вони зберігаються до бази даних після чого, його характеристики оновлюються коли користувач перемикається між вкладками. Наприклад:

- Якщо користувач обирає спортивне завдання (біг, присідання тощо), у нього підвищується сила та витривалість. Так як дані зберігаються вони порівнюються кожен раз коли користувач натискає згенерувати завдання або коли повертається на попередню сторінку;
- Якщо користувач вивчає навчальний матеріал та проходить тести по нім, у нього зростає інтелект;
- За кожну активність користувач отримує досвід (XP), який накопичується у загальний лічильник. Після досягнення певного порогу XP (100 одиниць), відбувається підвищення рівня (Level Up);

Блок-схема на Рисунок 2.10 ілюструє механізм оновлення характеристик користувача, збереження та отримання даних, залежно від його дій у додатку.

Основна логіка побудована на події – виконання певної дії (вправа, навчання тощо), після якої вони зберігаються та запускається процес оновлення відповідних параметрів.

Основні етапи в блок-схемі:

1. Дія користувача
    a. Користувач обирає тип активності: спортивне завдання або навчальний модуль.
2. Перевірка типу активності
    a. Якщо обрана дія належить до категорії "Спорт", оновлюються фізичні характеристики (сила, витривалість), які аналізуються зміни кількості повторень вправ, що призводить до зміни фізичних характеристик.
    b. Якщо обрано "Навчання", аналізується статус виконаних тестів, що впливає на інтелект.
3. Оновлення досвіду
    a. За кожну активність нараховується XP — по 10 одиниць при покращенні результату, або 10 при зменшенні.
    b. Весь досвід накопичується в загальний лічильник.
4. Перевірка на підвищення рівня
    a. Якщо кількість досвіду досягла певного порогу, відбувається підвищення рівня користувача (Level Up).
    b. У разі недосягнення — користувач продовжує виконувати дії для накопичення досвіду.
5. Завершення процесу
    a. Оновлені характеристики зберігаються.
    b. Користувач може продовжити взаємодію або повернутись до головного меню.

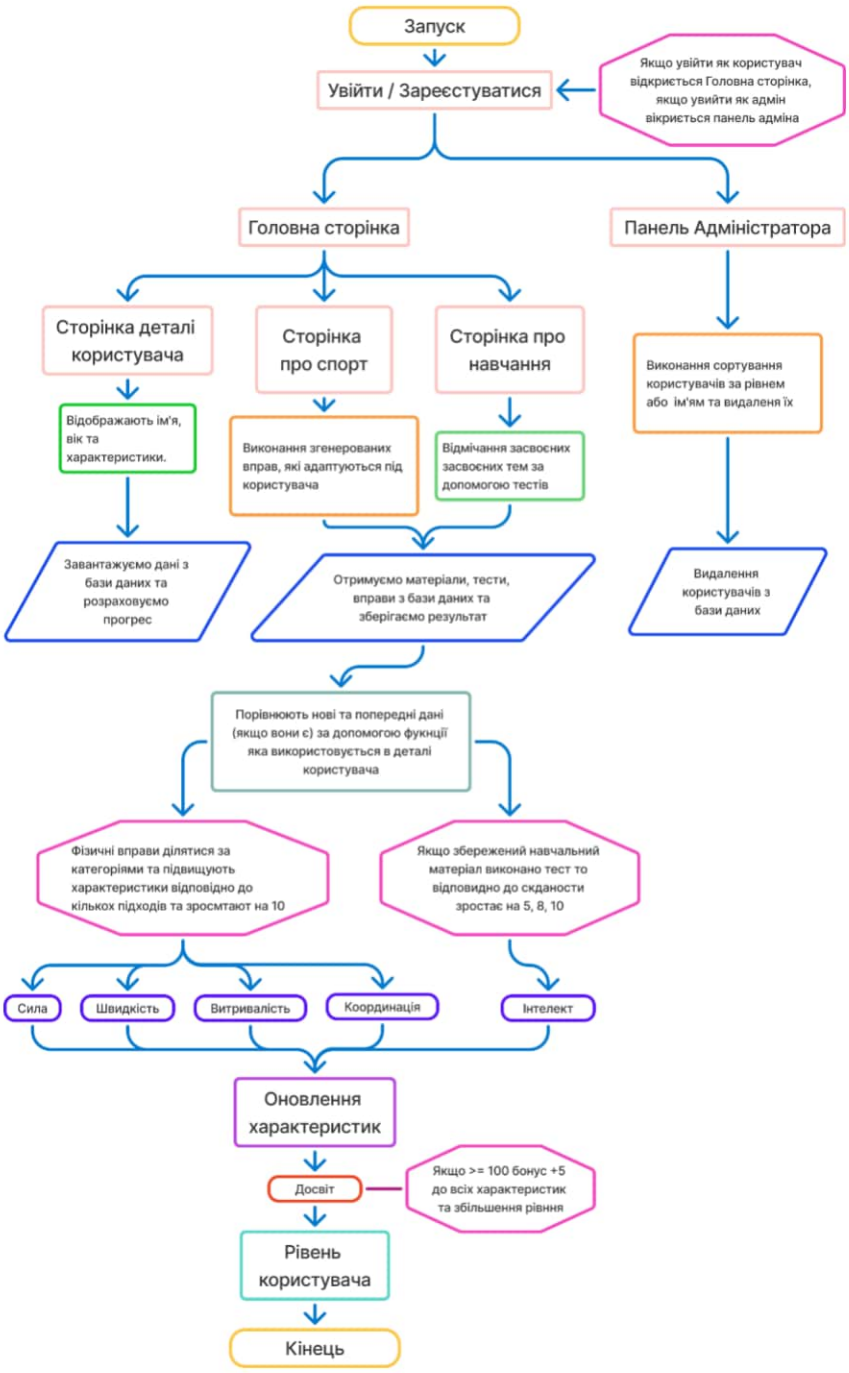


Рисунок 2.11 – Блок-схема оновлення характеристик користувача.

Ця блок-схема слугує візуальним представленням логіки гейміфікації у додатку. Вона відображає автоматизовану систему зростання характеристик і рівня користувача на основі активностей, що мотивує до постійного використання додатку. Механізм простий, але ефективний – надає користувачу відчуття прогресу, що є ключовим фактором у розвитку звички до самовдосконалення.

2.12. Відстеження засвоєних навчальних тем

2.12.1. Функціональність відстеження навчальних тем

Функціональність відстеження навчального прогресу реалізована через збереження даних у базі для кожного користувача. Прогрес визначається за такими критеріями, як проходження тестів, відкриття матеріалів та кількість успішно завершених блоків у кожній темі.

Для кожної теми підраховується кількість пройдених блоків (максимум – 3), і ця інформація відображається безпосередньо на кнопці теми у форматі: "Назва теми (2/3)". Це дозволяє користувачу бачити, скільки частин матеріалу вже опрацьовано. Блоки навчального матеріалу відкриваються поступово. Кожен наступний блок стає доступним лише після успішного завершення попереднього.

Якщо попередній блок не пройдено, виводиться повідомлення: «Блок X буде доступний після проходження попереднього тесту». Після проходження тесту зберігається результат (кількість балів, статус проходження), оновлюється інформація про відкриті матеріали та стан теми. Також викликається функція оновлення характеристик користувача, що дозволяє програмі адаптувати навчальний процес відповідно до прогресу. Цей підхід подібний до систем, які використовуються в Duolingo, Coursera або Udemy, де користувач бачить індикатори прогресу й доступність наступних частин. У нашій програмі усе реалізовано локально, без складної навігації чи реєстрації, що забезпечує простоту та зручність у використанні.

2.12.2. Логіка зміни статусу теми

Статус теми змінюється автоматично на основі результатів проходження тестів і відкриття навчальних блоків. Прогрес користувача зберігається у базі даних, яка відображає, які блоки теми були успішно опрацьовані.

Кожен навчальний блок доступний лише після проходження попереднього, що забезпечує послідовність опрацювання матеріалу. Якщо

користувач не пройшов тест або повторно виконує матеріал, статус теми оновлюється відповідно до нових результатів.

Таким чином користувач має можливість самостійно коригувати свій прогрес, перепроходити тести або повторно переглядати матеріал для кращого засвоєння. Прогрес відображається через індикатори на рівні тем та блоків, що дозволяє швидко орієнтуватися у рівні засвоєння.

Цей підхід дає свободу у навчанні, не нав’язуючи жорстку послідовність, та допомагає користувачам будь-якого віку гнучко управляти власним навчальним процесом. Завдяки цьому засвоєння матеріалу стає більш усвідомленим, а процес навчання — ефективним і зручним. тем. У результаті засвоєння знань стає більш усвідомленим і послідовним.

2.13. Система персональних цілей

2.13.1. Призначення системи персональних цілей

Система персональних цілей реалізована як окремий розділ інтерфейсу користувача, де кожен користувач може керувати власними завданнями щодо саморозвитку. Основне призначення системи – надати простий і зручний спосіб фіксувати цілі, стежити за їх виконанням і формувати власну траєкторію розвитку. Основний функціонал:

- Створення нової цілі: Користувач може натиснути кнопку «Встановити ціль», після чого з’являється вікно введення тексту мети. Після підтвердження нова ціль додається до загального списку;
- Відображення списку цілей: Усі цілі виводяться на екран у вигляді списку. Для кожної цілі вказано її текст, кнопка видалення та чекбокс для позначення виконання;
- Позначення цілі як виконаної: За допомогою чекбоксу користувач може відмітити ціль як виконану. У такому випадку текст цілі змінює стиль – стає перекресленим і зеленого кольору.
- Видалення цілей: Користувач може видалити будь-яку ціль натисканням на відповідну кнопку.

- Відображення верхньому правому кутку відображається діаграма характеристик, щоб було краще обрати свій напрямок або ціль.

Усі дані зберігаються до бази даних, завдяки чому список цілей зберігається між сесіями.

Також система радить обрати один із трьох напрямів розвитку, кожен із яких містить короткі рекомендації:

- Фізичний розвиток: рекомендації включають заняття спортом, правильне харчування, режим сну;
- Інтелектуальний розвиток: користувачу пропонується читати книги, проходити онлайн-курси, вивчати іноземні мови;
- Емоційне здоров'я: наголошується на важливості медитації, спілкування з близькими та психологічної підтримки.

Або просто дивлячись на характеристики у вигляді діаграми, щоб самостійно обрати свій напрямок.

Психологічний ефект. Просте позначення завдання як виконаного створює ефект досягнення, який позитивно впливає на мотивацію користувача. Такий підхід використовується в популярних застосунках, як-от Habitica або Google Tasks, де навіть мінімальні дії користувача дають відчуття прогресу.

Загалом, ця система сприяє формуванню навичок самопланування, самодисципліни й відповідальності, а також слугує візуальним трекером особистого розвитку.

2.13.2. Інтерактивне керування цілями

У програмі реалізовано повноцінний функціонал для керування персональними цілями користувача в інтерактивному режимі. Користувач має змогу створювати нові цілі, переглядати, редагувати їхній стан, а також видаляти за потреби. Це дозволяє вести власний список завдань і планів без використання сторонніх сервісів.

Для створення нової цілі достатньо натиснути кнопку «Встановити ціль», після чого відкривається спеціальне діалогове вікно з текстовим полем. Користувач вводить текст своєї мети та натискає кнопку «Зберегти» – нова ціль одразу з’являється у списку. Завдяки простій формі введення та зручному інтерфейсу, процес постановки цілей стає швидким та інтуїтивно зрозумілим.

Кожна ціль у списку супроводжується двома інтерактивними елементами:

- Чекбоксом для позначення виконуваності. Якщо користувач відзначає мету як виконану, вона візуально змінюється – текст перекреслюється та підсвічується зеленим кольором. Це дозволяє легко відрізнити завершені завдання від тих, що ще потребують уваги.
- Кнопкою видалення у вигляді іконки смітника. При її натисканні мета одразу видаляється зі списку, а зміни зберігаються у локальному файлі.

Усі дії користувача з цілями (створення, оновлення, видалення, виконання) зберігаються в базі даних, що забезпечує збереження даних між сесіями без потреби в підключенні до інтернету.

Окрім керування списком, користувач може обрати напрямок розвитку особистості – фізичний, інтелектуальний або емоційний. Кожен напрям супроводжується короткими рекомендаціями, що мотивують до формування корисних звичок і самовдосконалення.

Таким чином, цей розділ програми не лише допомагає формалізувати та структурувати персональні цілі користувача, а й сприяє систематичному підходу до особистісного зростання через наочне планування та контроль виконання.

2.14. Інтеграція функцій у загальний інтерфейс

2.14.1. Додаткові функції у головному вікні програми

Щоб зробити роботу з програмою максимально зручною, всі нові функції були інтегровані в загальний інтерфейс.

- Міні-чат-бот представлений на головній сторінці, доступний у будь-який момент;
- Оновлення характеристик відбувається автоматично для цього йому потрібно натиснути на кнопку оновити;
- Навчальні теми позначаються засвоєними безпосередньо у вікні перегляду матеріалу;
- Список персональних цілей інтегровано в особистий профіль користувача.

Інтеграція всіх додаткових функцій у головний інтерфейс забезпечує єдину, зручну точку доступу до всіх інструментів програми. Це означає, що користувачеві не потрібно шукати функції у складному меню або перемикатися між кількома вікнами – усе необхідне розташоване на одному екрані або у кількох вкладках.

Міні-чат-бот знаходиться на головній сторінці у вигляді окремого модуля, який завжди "під рукою". Завдяки цьому користувач може спілкуватися з ботом у фоновому режимі – наприклад, під час вивчення тем чи встановлення цілей. Бот миттєво реагує на запити і виконує роль особистого помічника у будь-який момент.

Блок оновлення характеристик винесено у верхню частину сторінки профілю, поруч із особистою інформацією користувача. Це дозволяє швидко відстежувати зміни параметрів та оновлювати їх у зручний момент, без додаткових переходів.

Система персональних цілей інтегрована як частина вкладки профілю, що дозволяє не лише бачити власний прогрес, але й сформувати чіткий план на майбутнє. Кожна мета відображається у вигляді окремого блоку з

можливістю взаємодії – зміни статусу або видалення.

Навчальні теми мають індикатори статусу прямо у вікні перегляду матеріалу, що робить процес навчання більш гнучким і контрольованим. Користувач бачить не лише тему, а й те, чи була вона вже опрацьована, що зменшує дублювання і допомагає краще структурувати знання.

Таким чином, інтерфейс забезпечує зручність, адаптивність і прозорість ключові чинники, що впливають на ефективність роботи з додатком.

2.15. Висновок до Розділу

У цьому розділі було сформульовано основну мету та завдання кваліфікаційної роботи – створення інформаційної системи, що виступає цифровим асистентом для підтримки особистого розвитку користувача. Описано функціональні та нефункціональні вимоги до системи, вимоги до безпеки, сумісності, а також обґрунтовано вибір сучасних технологій, що використовувалися при розробці.

Розглянуто архітектуру, структуру коду та ключові алгоритми функціонування програми. Окрему увагу приділено використанню PostgreSQL для збереження даних, що забезпечує масштабованість та гнучкість системи.

Реалізовані функції – міні-чат-бот, система постановки та досягнення персональних цілей, автоматичне оновлення характеристик, відстеження засвоєних навчальних тем, суттєво підвищують мотивацію користувача до саморозвитку. Інтерфейс програми реалізовано з використанням бібліотеки Flet, що забезпечує інтуїтивність, зручність і легкість у взаємодії.

Завдяки продуманій структурі коду, реалізована система легко адаптується під нові сценарії використання та підтримує розширення функціональності у майбутньому. Таким чином, створений програмний засіб є ефективним, гнучким і персоналізованим інструментом, що сприяє досягненню цілей користувача в освітньому, фізичному напрямах.

РОЗДІЛ 3.
ТЕСТУВАННЯ ТА ВАЛІДАЦІЯ ПРОГРАМИ

У цьому розділі розглянуто процес тестування та валідації розробленої програми. Основна мета тестування полягає у виявленні можливих помилок на різних етапах розробки, перевірці коректності роботи всіх основних функцій, а також забезпеченні стабільної та передбачуваної роботи системи у реальних умовах експлуатації. Проведення тестування дозволяє виявити слабкі місця у реалізації, забезпечити високий рівень якості програми та підвищити довіру кінцевих користувачів до розробленої системи.

3.1. Методологія тестування

Тестування програмного забезпечення – це критично важлива частина процесу розробки, яка дозволяє виявити помилки на ранніх етапах та забезпечити надійність роботи програми. У даному випадку було обрано метод тестування «чорної скриньки», що дозволяє перевірити роботу програми з точки зору користувача, не вдаючись до аналізу коду.

Метод «чорної скриньки» передбачає перевірку поведінки системи без знання її внутрішньої структури або коду. Тестувальник взаємодіє із програмою лише через інтерфейс і оцінює результати відповідно до заданих очікувань.

Завдяки використанню цього підходу вдалося зосередитись на тестуванні функціональності, інтерфейсу, швидкодії та взаємодії модулів, що дало змогу виявити і усунути низьку потенційних проблем.

Було виконано такі види тестування:

1. Функціональне тестування – перевірка роботи всіх основних функцій програми.
2. Тестування користувацького інтерфейсу – оцінка зручності та коректності відображення елементів.
3. Тестування продуктивності – аналіз швидкодії програми при обробці великої кількості даних.

4. Модульне тестування – перевірка окремих модулів програми.

5. Інтеграційне тестування – перевірка взаємодії між різними компонентами системи.

Застосування комплексного підходу дозволило не лише підтвердити правильність реалізації функціональності, а й виявити потенційні слабкі місця для подальшого удосконалення системи.

3.2. Функціональне тестування

Функціональне тестування охоплює перевірку основних можливостей програми, які безпосередньо використовуються кінцевим користувачем. Такий підхід дозволяє переконатися в коректності реалізації кожної з функцій та відповідності їх очікуваній поведінці.

Під час тестування особлива увага приділялась стабільності роботи програми та збереженню даних при взаємодії з різними функціональними модулями. Оскільки основна мета розробки – забезпечення комфортного користувацького досвіду, було важливо перевірити, чи кожен функціональний компонент працює надійно, логічно та без збоїв у різних сценаріях використання.

Функціональне тестування охоплювало всі основні можливості програми, включаючи:

- Взаємодію з міні-чат-ботом.
- Оновлення характеристик користувача відповідно до його дій.
- Позначення навчальних тем як "засвоєних".
- Роботу з персональними цілями (додавання, видалення, відзначення виконаних цілей).
- Збереження та зчитування даних з бази даних.

Для кожної з функцій було підготовлено сценарії тестування, які охоплювали як стандартні дії користувача, так і потенційно неочікувану поведінку. Завдяки цьому вдалося підтвердити правильність роботи системи при різних навантаженнях і в різноманітних умовах.

3.2.1. Тестування міні-чат-бота

Чат-бот є інтерактивним інструментом для комунікації з користувачем, тому його стабільність і правильне розпізнавання команд – ключовий аспект функціональної складової програми. Його завдання – надати швидкий зворотний зв’язок, допомагає користувачу у навігації програмою, а також надати інформацію чи розважальні елементи (наприклад, жарт або цитату).

Було перевірено широкий спектр запитів, включаючи як стандартні команди, так і введення невідомих фраз. Також перевірялась здатність чат-бота очищати історію повідомлень, відповідати на питання про поточний час і дату, а також реагувати на привітання або запити користувача.

Це дозволило оцінити, наскільки система орієнтована на користувача, та забезпечити надання відповідного зворотного зв'язку.

Були перевірені основні команди чат-бота ("привіт", "як справи?", "жарт", "цитата", "час", "дата", "очистити", тощо). Результати тестування:

- Усі команди розпізнаються коректно незалежно від регістру введення.

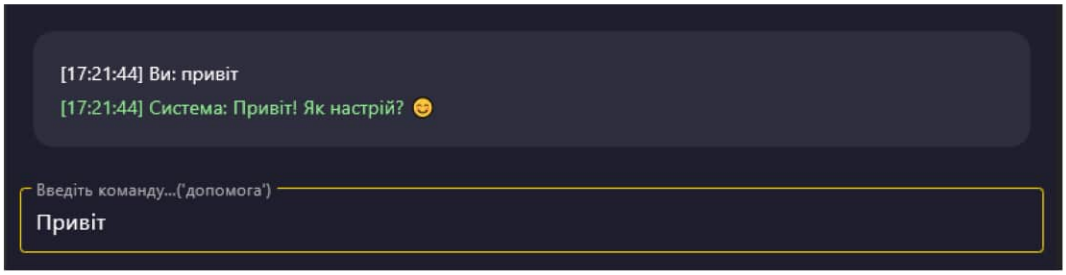


Рисунок 3.12 – Тестування команд.

- У разі введення невідомої команди бот пропонує список доступних команд для орієнтації користувача.

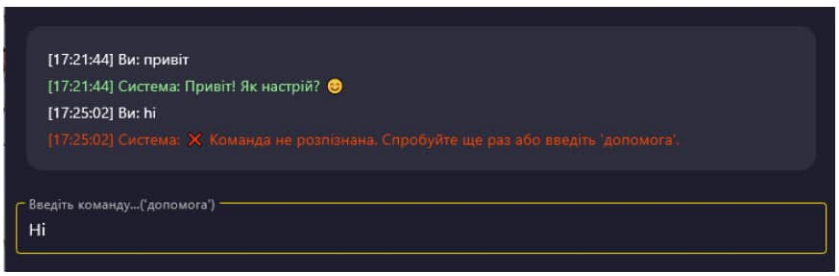


Рисунок 3.13 – Тестування на неправильні команд.

- Очищення історії чату працює без збоїв та миттєво реагує на запит.

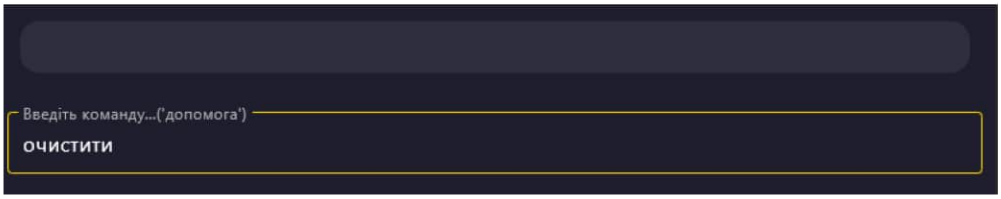


Рисунок 3.14 – Тестування команди очищення.

- Чат-бот працює стабільно навіть при великій кількості запитів підряд.

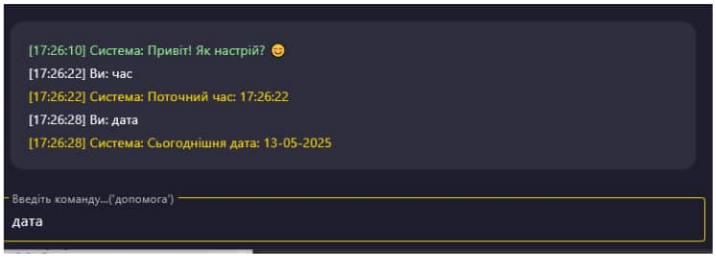


Рисунок 3.15 – Тестування написання декількох команд підряд.

3.2.2. Тестування оновлення характеристик користувача

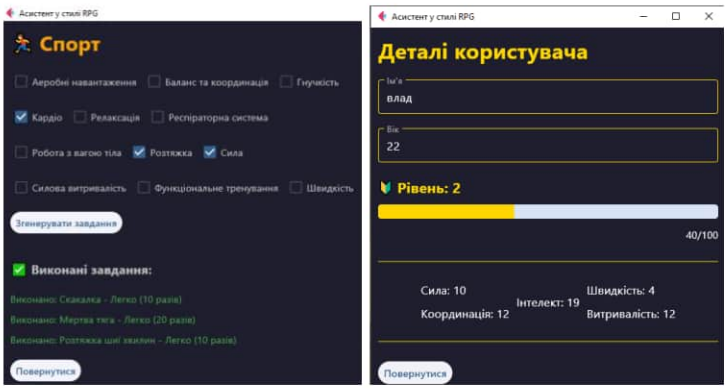
Оновлення характеристик – це важливий елемент персоналізації системи. Цей механізм дозволяє програмі адаптуватися до дій користувача, створюючи ефект прогресу та розвитку. Зміни показників повинні чітко відповідати діям користувача, і будь-яка неточність у реалізації може призвести до втрати довіри до системи.

Тестування проводилось для перевірки коректності нарахування досвіду, підвищення рівня, зміни характеристик (сила, витривалість, швидкість, тощо) у відповідь на різні дії користувача. Крім того, перевірялась стійкість алгоритмів до багаторазових послідовних змін, щоб виключити ризик накопичення некоректних значень або перевищення допустимих меж.

Були проведені тести на зміну рівня, досвіду та характеристик користувача.

Результати:

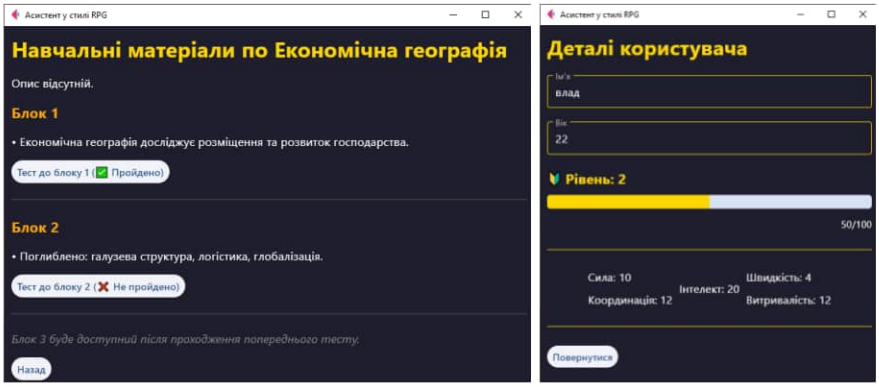
- При виконанні спортивних вправ відповідні показники (сила, витривалість, швидкість) збільшуються згідно з логікою програми.



а) б)

Рисунок 3.16 – Тестування зміни даних при виконання вправ: а) виконани вправи; б) зміна характеристик відповідно вправам.

- При вивченні навчального матеріалу підвищується інтелект користувача.



а) б)

Рисунок 3.17 – Тестування зміни даних при засвоєння матеріалів: а) відмічаємо що материал був засвоєний; б) підвищується інтелект користувача.

- Після накопичення достатньої кількості в досвіду при виконання вправ та засвоєних матеріалі відбувається підвищення рівня користувача.

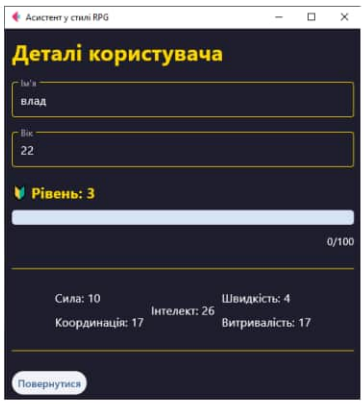
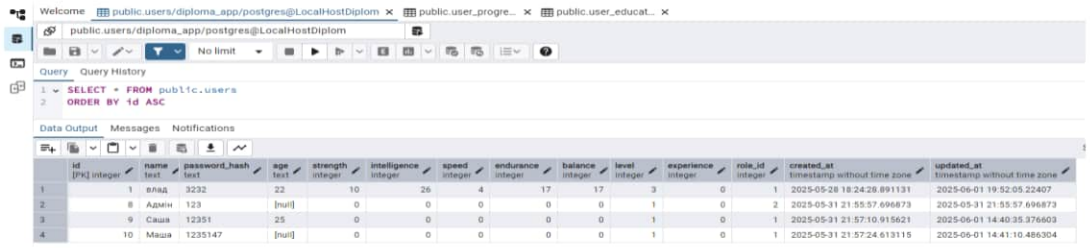
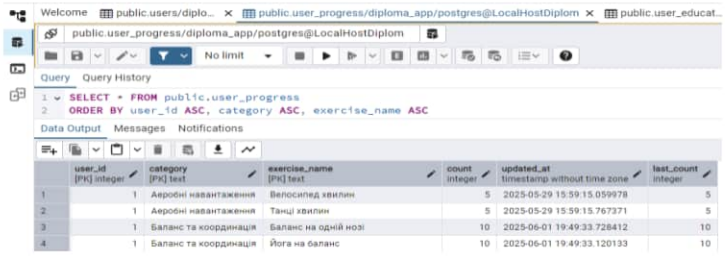


Рисунок 3.18 – Перевірка підвищення рівня після накопичення достатнього досвіду.

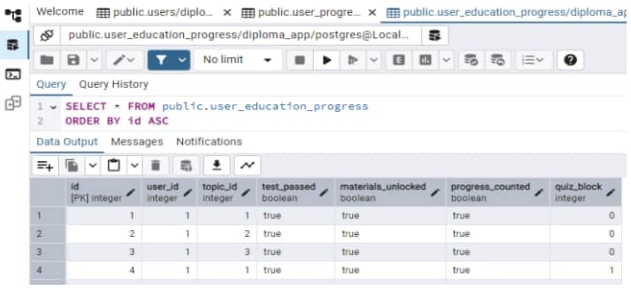
- Усі зміни зберігаються в базі даних у вигляді таблиць які коректно зчитуються при повторному запуску програми.



а)



б)



с)

Рисунок 3.19 – Перевірка зберігання даних в базі даних: а) збережені дані користувачів та адмінів б) нинішня і попередня інформація вправ; с) збережені інформація пройдених тестів.

3.2.3. Тестування позначення навчальних тем

Ця функціональність забезпечує фіксацію прогресу користувача у навчанні, що є важливою складовою персоналізованого підходу до розвитку. Вона відслідковує, які теми і блоки були успішно опрацьовані, та відображає цей стан у вигляді логічних статусів у базі даних.

Тестування було спрямоване на перевірку коректної взаємодії між інтерфейсом та внутрішньою логікою збереження і оновлення статусів. Особлива увага приділялася тому, чи відображається зміна прогресу користувача одразу після проходження тесту або опрацювання блоку, а також чи коректно зберігаються і відновлюються ці дані після перезапуску програми.

Було протестовано роботу з різними навчальними темами (наприклад, "Математика", "Географія", тощо) на різних етапах: початкове позначення пройдених блоків, зміна статусів при повторному проходженні матеріалу, та оновлення результатів тестів.

Результати тестування:

- Прогрес користувача миттєво оновлюється у системі після проходження тестів, що дає точний зворотний зв’язок про поточний стан засвоєння.

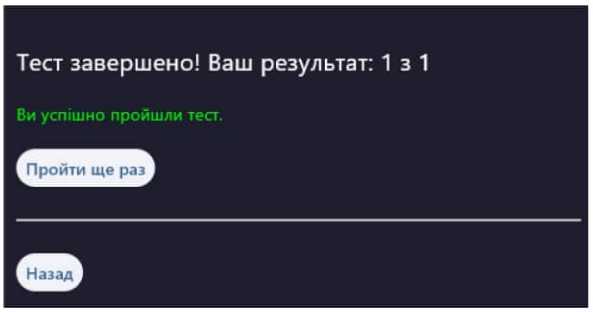


Рисунок 3.20 – Перевірка проходження теста.

- Статуси тем і блоків змінюються гнучно, що дозволяє користувачу повторно проходити матеріал для кращого засвоєння.

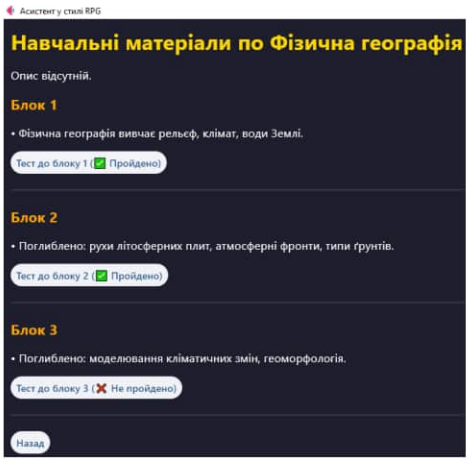


Рисунок 3.21 – Перевірка повторного прохождения теста.

- Дані про прогрес коректно зберігаються у базі даних і надійно відновлюються при наступному запуску програми.

22	22	1	5	true	true	true	2
23	23	1	14	true	true	true	1
24	24	1	14	true	true	true	2
25	25	1	12	true	true	true	1
26	26	1	12	false	true	false	2

Total rows: 26 Query complete 00:00:00.099

Рисунок 3.22 – Перевірка збереження тесту та його стану.

- Система демонструє стабільну поведінку незалежно від кількості тем та комбінації їх статусів.

3.2.4. Тестування персональних цілей

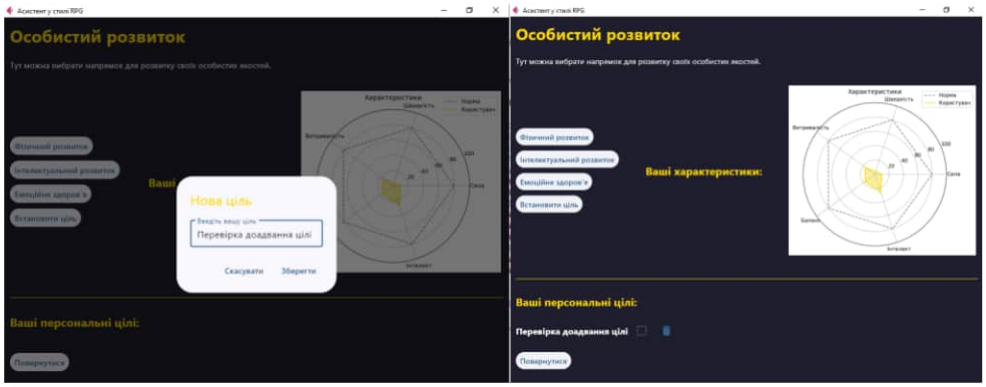
Робота з персональними цілями є частиною мотиваційної складової системи, яка сприяє формуванню цілеспрямованості користувача та дозволяє контролювати власний розвиток. Цей функціонал передбачає додавання нових цілей, їх видалення, а також відзначення як виконаних – із відповідною візуалізацією у вигляді підкреслення.

Під час тестування була оцінена як правильність роботи логіки, так і взаємодія з інтерфейсом користувача.

Особливу увагу приділено збереженню даних між сеансами – чи залишаються всі введені цілі після перезапуску програми, та чи зберігається інформація про їх виконання.

Результати тестування:

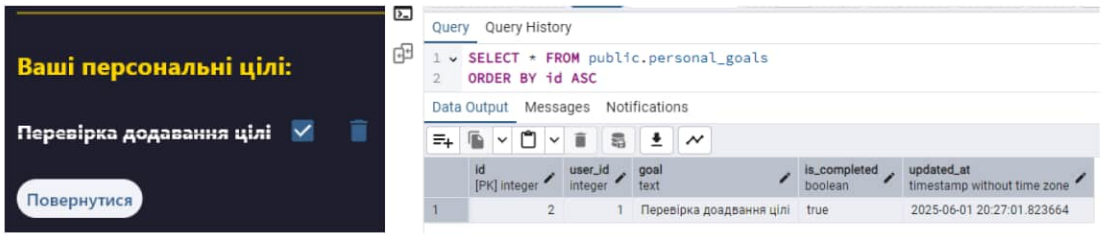
- Користувач може вільно додавати нові цілі з довільною назвою — без обмежень на кількість тексту.



а) б)

Рисунок 3.23 – Перевірка додавання нової цілі: а) процес додавання цілі; б) відображення доданої цілі.

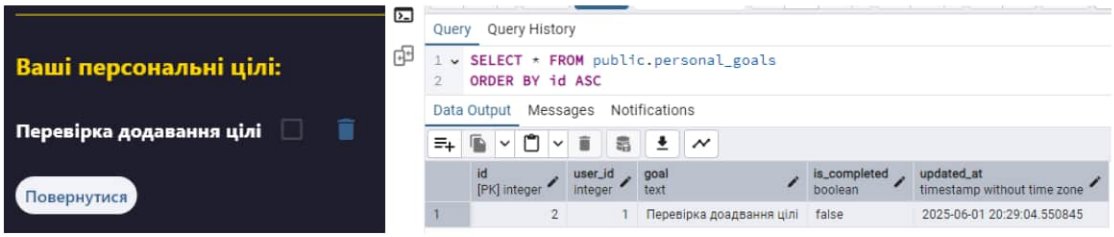
- Виконані цілі відмічаються галочкой та підкресленням, що надає зручний візуальний зворотний зв’язок.



а) б)

Рисунок 3.24 – Відмічена ціль як виконана: а) як відображається користувачу; б) як воно зберігається в таблиці.

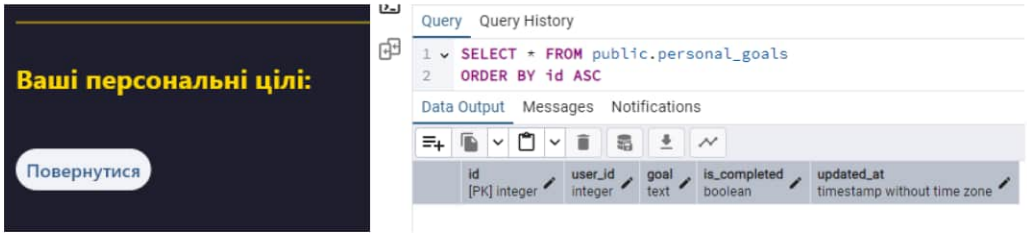
- Повторне натискання дозволяє скасувати виконання, що забезпечує гнучкість.



а) б)

Рисунок 3.25 – Повторне натискання на ціль: а) виконана ціль була скасована; б) як відображається в таблиці.

- Видалення цілей працює без помилок – ціль зникає з інтерфейсу і видаляється з бази даних.



а) б)

Рисунок 3.26 – Видалення цілі: а) віддалена ціль видаляється; б) ціль в файлі теж видаляється.

Таким чином, модуль управління персональними цілями працює надійно, не викликаючи труднощів у користувача, навіть при активному додаванні або зміні великої кількості записів.

3.3. Тестування користувацького інтерфейсу

Тестування інтерфейсу користувача дозволяє переконатися, що всі елементи відображаються правильно, функціонують без помилок та забезпечують зручну взаємодію з системою. Оскільки інтерфейс є основною точкою контакту між користувачем і програмним забезпеченням, особлива увага приділялась як зовнішньому вигляду, так і логіці поведінки елементів.

Було проведено перевірку на різних розмірах екрану, а також з різними наборами вхідних даних, щоб впевнитися у відсутності візуальних або логічних помилок.

Під час тестування перевірялися такі аспекти:

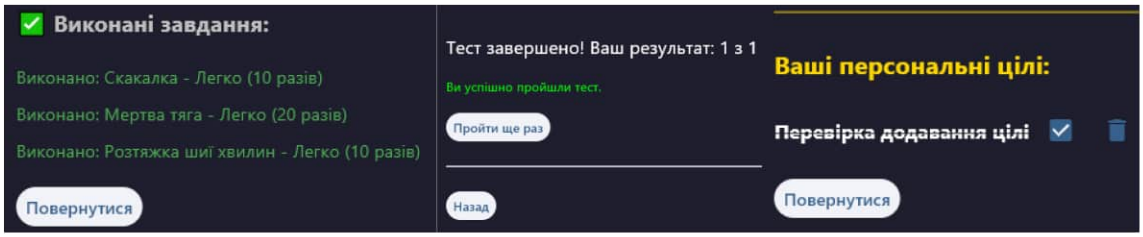
- Відображення всіх кнопок, підписів, текстових полів та інших елементів інтерфейсу;
- Коректна реакція на взаємодію (натискання, наведення курсору);
- Адаптивність інтерфейсу при зміні розміру вікна програми;
- Чітке логічне розташування елементів (зрозуміла структура, відсутність перевантаження або хаотичного розташування);

- Підтримка кольорового маркування важливих дій (наприклад, позначення засвоєних тем або виконаних цілей).

Особливу увагу було приділено елементам зворотного зв'язку: чи зручно користувачеві розуміти, що дія виконана.

Результати тестування:

- Всі елементи інтерфейсу відображаються стабільно без зсувів чи перекриттів.
- Візуальні зміни після дій користувача працюють коректно та забезпечують зрозумілий зворотний зв’язок.



а) б) с)

Рисунок 3.27 – Візуальний зворотній зв'язок: а) виконані вправи; б) проходження тесту; с) досягнута ціль.

- Навігація по розділах (спорт, навчання, цілі) є логічною та інтуїтивно зрозумілою.

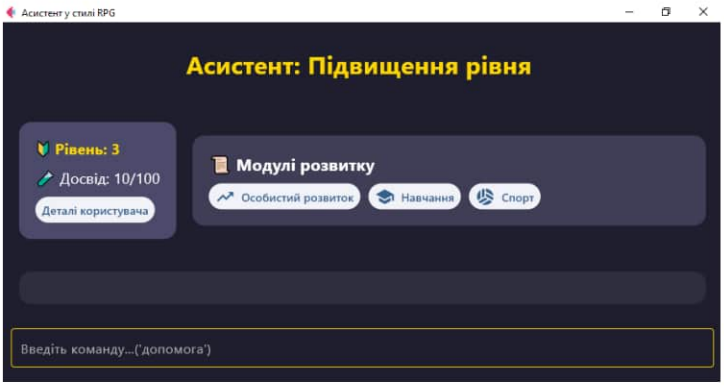


Рисунок 3.28 – Навігаційна на головній сторінці.

Таким чином, інтерфейс є зручним, стабільним та орієнтованим на кінцевого користувача, що позитивно впливає на загальне враження від роботи з програмою.

3.3.1. Перевірка відображення елементів

Були протестовані різні роздільні здатності екранів і режими вікон, щоб переконатися в адаптивності інтерфейсу. Графічні елементи залишаються на своїх місцях і не перекриваються. Оцінювалася коректність відображення кнопок, текстових полів та інших елементів UI. Також враховувались особливості сприйняття користувачем – розміри шрифтів, контрастність, доступність до навігаційних кнопок.

Додатково перевірялися такі аспекти:

- чи зберігається логічна структура при зміні масштабу;
- чи не виходять елементи за межі вікна.

Результати:

- Всі елементи відображаються коректно на різних розмірах екрана;
- Колірна схема та дизайн залишаються стабільними при зміні параметрів вікна;
- Відсутні візуальні дефекти, такі як накладання елементів або зміщення тексту;
- Зміна масштабу не впливає на функціональність або читабельність інтерфейсу.

3.3.2. Перевірка інтерактивності

Важливою частиною UI є чутливість до дій користувача. Кнопки та інші інтерактивні елементи повинні швидко реагувати на натискання, що було підтверджено в результаті тестування. Особлива увага приділялась стабільності обробки подій, уникненню подвійного спрацювання елементів, а також візуальному зворотному зв’язку.

Також було перевірено:

- коректність обробки натискань на кнопки у різних станах інтерфейсу (повноекранний/віконний режим);
- роботу з чекбоксами, списками та полями введення;

- реакцію інтерфейсу на швидкі та багаторазові дії користувача;
- наявність візуальних змін при взаємодії .

Результати:

- Кнопки реагують на кліки без затримок;
- Всі інтерактивні елементи працюють стабільно;
- Випадкові помилки або зависання відсутні;
- Користувач отримує миттєвий візуальний відгук на свої дії.

3.4. Тестування адміністративного інтерфейсу

Адміністративний інтерфейс реалізовано за допомогою бібліотеки Flet. Він надає можливість перегляду користувачів, сортування за іменем, відображення основних характеристик та видалення облікових записів із бази даних. Для тестування цієї частини інтерфейсу було перевірено всі основні функції адміністратора.

У процесі тестування було виконано такі дії:

- Завантаження списку користувачів: перевірено правильне відображення імен, рівня, досвіду та п’яти основних характеристик (сила, інтелект, швидкість, витривалість, баланс) для кожного користувача.
- Оновлення списку користувачів: протестовано сортування за іменем, а також правильне відображення максимум трьох користувачів одночасно відповідно до логіки відбору.
- Функція видалення користувача: перевірено, що після натискання кнопки «Видалити» відповідний запис видаляється з бази даних, список користувачів оновлюється, а користувач отримує візуальне підтвердження через SnackBar.

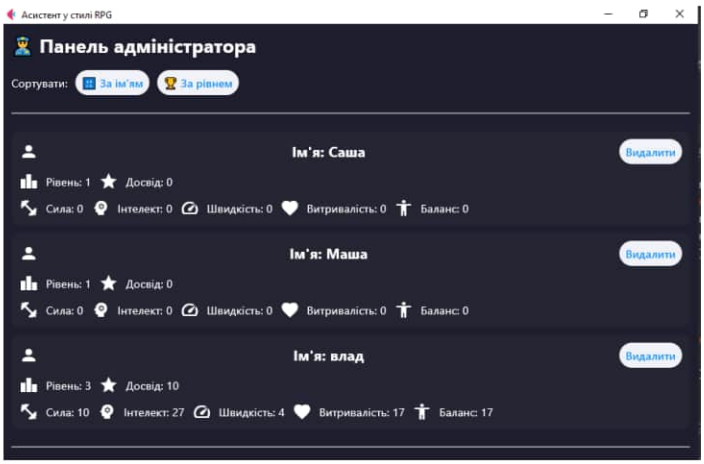


Рисунок 3.29 – Навігаційна панель адміністратора.

Тестування показало, що адміністративна панель працює стабільно, а всі дії виконуються коректно й синхронізовано з базою даних. Інтерфейс є зручним для використання та забезпечує базовий контроль над обліковими записами користувачів.

3.5. Тестування продуктивності

У рамках тестування продуктивності було перевірено стабільність роботи програми при збільшенні обсягу даних у базі. Хоча ручне масове введення інформації не передбачено, система активно взаємодіє з базою даних, де зберігаються дані про навчальні теми, характеристики користувача, його цілі та прогрес у навчанні.

На практиці було перевірено:

- стабільність роботи програми після додавання великої кількості персональних цілей;
- швидкість зчитування навчальних тем;
- коректність збереження великих обсягів даних;
- відсутність зависань під час виконання кількох дій поспіль.

Таблиця 3.9 – Підсумкові результати тестування продуктивності

№	Перевірений аспект	Результат
1.	Відкриття програми після значних змін у даних	Працює стабільно
2.	Зчитування тем з бази даних	Дані відображаються коректно
3.	Додавання великої кількості цілей	Жодних збоїв чи зависань
4.	Збереження змін у бази даних	Всі зміни зберігаються коректно
5.	Робота інтерфейсу при навантаженні	Програма реагує без затримок

3.6. Модульне тестування

Для забезпечення якісної перевірки внутрішньої логіки роботи окремих частин програми було проведено модульне тестування. Такий підхід дозволяє локалізувати помилки на ранніх етапах розробки, не залучаючи повністю всю систему.

Хоча основне тестування виконувалось вручну, також застосовувались елементи автоматизованого підходу: повторювані дії виконувались із заготовленими даними, що дозволяло пришвидшити перевірку стабільності та передбачуваності роботи окремих функцій.

Перевірені модулі:

- Функція збереження та завантаження даних: Тестувалося зчитування й запис інформації до баз даних. Під час тестів програма стабільно працювала навіть після, а структура збережених даних залишалася валідною;
- Функція зміни характеристик користувача: Перевірено коректність збільшення досвіду, рівня та особистих характеристик на

основі виконаних дій. Значення оновлювались згідно з логікою програми;

- Функція генерації відповідей чат-бота: Було протестовано різноманітні вхідні повідомлення, включно з помилковими або нестандартними запитами. Усі відповіді формувалися адекватно, з дотриманням визначених шаблонів.

Таблиця 3.10 – Результати модульного тестування.

№	Модуль	Стан виконання
1.	Збереження та завантаження даних	Працює стабільно
2.	Оновлення характеристик	Дані змінюються коректно
3.	Відповіді чат-бота	Відповідає шаблону

3.7. Інтеграційне тестування

На етапі інтеграційного тестування було перевірено, як окремі частини системи взаємодіють між собою. Такий підхід дозволяє виявити конфлікти між модулями, що не були помітні на етапі модульного тестування. Було протестовано взаємодію між основним інтерфейсом, модулями збереження/завантаження даних, логікою обробки подій, а також між різними сторінками застосунку, зокрема особистого розвитку, спортивних завдань та навчальних модулів.

Окрема увага приділялася перевірці збереження змін у базі даних, а також перевірці стабільності роботи інтерфейсу після переходу між сторінками, створення нових цілей і виконання дій користувача у довільному порядку.

Результати підтвердили правильну організацію зв’язків між компонентами та стабільну роботу системи.

Результати:

- Програма працює стабільно, незалежно від послідовності виконання дій;
- Взаємодія між компонентами відбувається без конфліктів;
- Дані коректно зберігаються та завантажуються при переході між сторінками;
- Події (натискання кнопок, введення тексту, вибір опцій) викликають очікувану реакцію у відповідних модулях.

3.8. Висновок до розділу

У цьому розділі було детально розглянуто процес тестування та валідації програми. Завдяки застосуванню комплексного підходу до перевірки, що включає функціональне, модульне, інтеграційне тестування, а також перевірку інтерфейсу та продуктивності, вдалося виявити і усунути всі виявлені недоліки.

Особливу увагу було приділено перевірці взаємодії між окремими модулями, збереженням користувацьких даних та коректній роботі з різними сценаріями використання. Усі ключові функції системи пройшли тестування без критичних збоїв.

Програма продемонструвала високу стабільність, коректну реакцію на дії користувача та відповідність функціоналу поставленим вимогам. Отримані результати підтверджують готовність програмного продукту до використання кінцевими користувачами.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи було розроблено інтерактивну інформаційну систему – персонального асистента, що сприяє навчальному та фізичному розвитку користувача. Система дозволяє встановлювати особисті цілі, виконувати завдання, відстежувати прогрес і взаємодіяти з чат-ботом для отримання порад, жартів або корисної інформації.

Програму реалізовано на основі сучасних технологій з акцентом на стабільність роботи, зручний інтерфейс та розширену структуру. Уся логіка проєкту розділена на окремі модулі, серед яких:

- main.py – головна точка запуску програми, маршрутизація між сторінками;
- auth_page.py – авторизація та реєстрація користувача;
- admin.py – панель адміністратора для керування контентом;
- user_details_page.py – оновлення характеристик, досвіду та рівня користувача;
- education_page.py – освітній модуль із темами та перевіркою засвоєння матеріалу;
- sport_page.py – фізичні вправи з адаптивним навантаженням та оцінкою складності;
- personal_development_page.py – постановка, перегляд і виконання персональних цілей;
- db.py – підключення та взаємодія з базою даних PostgreSQL;
- logic.py – логіка чат-бота: поради, жартівливі відповіді, підказки;
- ui_components.py – графічні елементи інтерфейсу на базі бібліотеки Flet.

Для реалізації використано базу даних PostgreSQL. Це забезпечує вищий рівень надійності, дозволяє ефективно організувати зв’язки між таблицями та масштабувати об'єкт під майбутні сценарії використання.

Основні функції системи охоплюють: навчальні модулі з темами, фізичні вправи з адаптивним навантаженням, встановлення особистих цілей, чат-бот із гейміфікаційними елементами, динамічне оновлення рівня та характеристик користувача. Такий підхід створює ефективне середовище для саморозвитку.

Завдяки бібліотеці Flet реалізовано простий, але функціональний інтерфейс, що не потребує додаткових налаштувань. Усі функціональні компоненти пройшли тестування. Перевірка підтвердила:

- стабільну роботу системи;
- коректне оновлення даних користувача;
- відповідність між введеними діями та результатами;
- відсутність критичних помилок;
- зручність інтерфейсу.

Система демонструє високу надійність у роботі та чітку взаємодію між модулями. Завдяки структурованості коду вона легко піддається модифікації і може бути розширена у майбутньому. Можливості подальшого розвитку:

- розширення бази знань та вправ;
- глибока інтеграція з нейромережею для персоналізації;
- візуалізація прогресу у вигляді графіків;
- вдосконалення інтелектуальних функцій чат-бота.

Інтеграція системи саморозвитку з гейміфікацією та чат-ботом довела свою ефективність у стимулюванні навчання й фізичної активності. Створена програма є готовим функціональним рішенням для користувачів, що прагнуть покращити навички та досягати цілей у зручній, інтерактивній формі.

Завдяки модульній архітектурі програму можна легко адаптувати під нові вимоги та сценарії. Таким чином, усі поставлені у кваліфікаційній роботі завдання виконано повністю, а розроблена система є перспективним інструментом для подальшого розвитку та вдосконалення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Python Software Foundation. Python Documentation. [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/ (дата звернення: 11.04.2025).

2. Flet. Офіційна документація. [Електронний ресурс]. – Режим доступу: https://flet.dev/ (дата звернення: 11.04.2025).

3. YouTube. Відео уроки з Python. [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=34Rp6KVGIEM&list=PLDyJYA6aTY1lPWXBPk0gw6gR8fEtPDGKa (дата звернення: 11.04.2025).

4. YouTube. Відео уроки з Flet. [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=OcrEMgx7OjE&list=PL0lO_mIqDDFVZr9lLryYHSbAbrn3YJGbE (дата звернення: 11.04.2025).

5. Python Software Foundation. Документація модуля json. [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/library/json.html (дата звернення: 11.04.2025).

6. Python Software Foundation. Робота з файлами в Python. [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/tutorial/inputoutput.html (дата звернення: 11.04.2025).

7. PostgreSQL. Офіційна документація. [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/ (дата звернення: 11.04.2025).

8. HelpCrunch. Стаття: Що таке чат-бот? [Електронний ресурс]. – Режим доступу: https://helpcrunch.com/blog/uk/shcho-take-chat-bot/ (дата звернення: 11.04.2025).

9. GitHub. Flet Examples – Репозиторій прикладів. [Електронний ресурс]. – Режим доступу: https://github.com/flet-dev/examples (дата звернення: 11.04.2025).

10. Flet: Building Flutter-like web, desktop and mobile apps in Python. GitHub-репозиторій. [Електронний ресурс]. – Режим доступу:

https://github.com/flet-dev/flet (дата звернення: 11.04.2025).

11. Real Python. Python Tips & Tricks. [Електронний ресурс]. – Режим доступу: https://realpython.com/ (дата звернення: 11.04.2025).

12. Ідеї щодо навчальних функцій були адаптовані з мобільних застосунків для вивчення предметів, таких як Duolingo, Quizlet тощо.

13. Ідеї щодо фізичних вправ були натхненні функціоналом мобільного застосунку “Тренування для дому” (Home Workout).

14. Особистий досвід користування Python та Flet під час розробки застосунку.

15. Консультації та поради, отримані під час спілкування з ChatGPT. [Електронний ресурс]. – Режим доступу: https://chat.openai.com/ (дата звернення: 11.04.2025).

16. StriveCloud. Strava Gamification Playbook. [Електронний ресурс]. – Режим доступу: https://strivecloud.io/play/strava/ (дата звернення: 02.05.2025).

17. StriveCloud. Khan Academy Gamification Playbook. [Електронний ресурс]. – Режим доступу: https://strivecloud.io/play/khan-academy-gamification-playbook/ (дата звернення: 02.05.2025).

18. Duolingo Blog. Introducing Adventures, a brand-new gamified learning experience. [Електронний ресурс]. – Режим доступу: https://blog.duolingo.com/adventures/ (дата звернення: 02.05.2025).

19. Sailer, M., Hense, J., Mayr, S., & Mandl, H. (2017). How gamification motivates: An experimental study of the effects of specific game design elements on psychological need satisfaction. Computers in Human Behavior, 69, 371–380. https://www.sciencedirect.com/science/article/pii/S074756321630855X

20. Deterding, S., Dixon, D., Khaled, R., & Nacke, L. (2011). From game design elements to gamefulness: Defining "gamification". In Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments (pp. 9–15). https://dl.acm.org/doi/10.1145/2181037.2181040

21. Gartner. (2023). What's New in Artificial Intelligence from the 2023 Gartner Hype Cycle. https://www.gartner.com/en/articles/what-s-new-in-artificial-intelligence-from-the-2023-gartner-hype-cycle

22. Duhigg, C. (2012). The Power of Habit: Why We Do What We Do in Life and Business. Random House. https://charlesduhigg.com/the-power-of-habit/

ДОДАТОК А. ФРАГМЕНТ ЛІСТИНГУ ПРОГРАМИ

Файл main.py

```
# Імпортуємо необхідні модулі з бібліотеки Flet і власні сторінки та компоненти
import flet as ft
from pages.admin import admin_page  # Сторінка адміністратора
from pages.user_details_page import user_details_page, update_characteristics
# Сторінка деталей користувача та функція оновлення характеристик
from pages.personal_development_page import personal_development_page  # Сторінка особистого розвитку
from pages.education_page import education_page  # Сторінка навчання
from pages.sport_page import sport_page  # Сторінка спорту
from ui_components import create_header, create_user_status, create_log_panel, create_user_input  # UI-компоненти
from logic import process_command  # Обробка команд користувача
from db import load_user_data, is_admin  # Завантаження даних користувача та перевірка на адміністратора
from pages.auth_page import get_auth_page  # Сторінка авторизації


# Основна сторінка програми після входу користувача
def main_page(page, max_log_length, user_status, log_panel, user_id):
    user_data = load_user_data(user_id)  # Завантажуємо дані користувача з бази
    update_characteristics(user_id)  # Оновлюємо характеристики користувача

    # Функції переходу до відповідних сторінок
    def navigate_to_personal_development(_):  # Перехід на сторінку особистого розвитку
        personal_development_page(page, lambda: main_page(page, max_log_length, user_status, log_panel, user_id),
                                  user_id)

    def navigate_to_education(_):  # Перехід на сторінку навчання
        education_page(page, lambda: main_page(page, max_log_length, user_status, log_panel, user_id), user_id)

    def navigate_to_sport(_):  # Перехід на сторінку спорту
        sport_page(page, lambda: main_page(page, max_log_length, user_status, log_panel, user_id), user_id)

    def navigate_to_user_details():  # Перехід на сторінку деталей користувача
        user_details_page(page, lambda: main_page(page, max_log_length, user_status, log_panel, user_id),
                          user_data)

    # Створення заголовка програми
    header = create_header()

    # Створення статусу користувача (наприклад, ім’я, рівень тощо)
    user_status = create_user_status(user_id, navigate_to_user_details)
```

```
    # Блок з кнопками для переходу до модулів розвитку
    module_panel = ft.Container(
        content=ft.Column(
            [
                ft.Text("□ Модулі розвитку", size=20, color="#FFFFFF",
weight=ft.FontWeight.BOLD),  # Назва модуля
                ft.Row(
                    [
                        ft.ElevatedButton("Особистий розвиток",
icon=ft.icons.TRENDING_UP,

on_click=navigate_to_personal_development),
                        ft.ElevatedButton("Навчання", icon=ft.icons.SCHOOL,
on_click=navigate_to_education),
                        ft.ElevatedButton("Спорт",
icon=ft.icons.SPORTS_VOLLEYBALL, on_click=navigate_to_sport),
                    ],
                    alignment=ft.MainAxisAlignment.SPACE_EVENLY,
                    wrap=True,
                ),
            ],
            spacing=10,
        ),
        padding=20,
        margin=10,
        bgcolor="#3E3E56",  # Темне тло
        border_radius=15,  # Закруглені кути
        expand=True,  # Розширення блоку по ширині
    )

    # Поле введення команд користувачем (наприклад, "зберегти", "статус")
    user_input = create_user_input(
        lambda command: process_command(command, log_panel, max_log_length)
    )

    # Очищення сторінки та додавання нових елементів
    page.clean()
    page.add(
        ft.Column(
            [
                header,  # Додаємо заголовок
                ft.Row([user_status, module_panel],
alignment=ft.MainAxisAlignment.SPACE_BETWEEN),  # Статус + модулі
                ft.Row([log_panel], alignment=ft.MainAxisAlignment.CENTER),
# Панель логів (журналу подій)
                user_input,  # Поле вводу команди
            ],
            spacing=20,
            expand=True,
        )
    )
```

```
# Головна функція запуску додатку
def main(page: ft.Page):
    # Налаштування вікна програми
    page.title = "Асистент у стилі RPG"
    page.theme_mode = "light"  # Світла тема
    page.padding = 10
    page.bgcolor = "#1E1E2E"  # Темний фон
    page.window_width = 800
    page.window_height = 550
    page.window_min_width = 510
    page.window_min_height = 500
    page.window_max_width = 900
    page.window_max_height = 800

    max_log_length = 5  # Максимальна кількість рядків у журналі подій
    log_panel = create_log_panel()  # Створюємо панель журналу

    # Колбек після успішного входу
    def on_login_success(user_id):
        page.clean()

        if is_admin(user_id):  # Якщо це адміністратор — відкриваємо
адмінську сторінку
            admin_page(page)
        else:  # Інакше — головну сторінку користувача
            main_page(page, max_log_length, None, log_panel, user_id)

    # Початкова сторінка авторизації
    page.clean()
    page.add(get_auth_page(on_login_success))
    page.update()


# Запуск додатку
ft.app(target=main)
```

Файл logic.py

```
import flet as ft
from datetime import datetime
import random

# Список жартів
jokes = [
    "Чому програмісти не люблять природу? — Бо там багато багів.",
    "Я б розповів ще один жарт про баги, але він іноді не працює.",
    "Як називається фобія видалення коду? — Delete-обія."
]

# Список цитат
quotes = [
```

```
    "«Єдиний спосіб зробити велику роботу — це любити те, що ви робите.» — Стів Джобс",
    "«Навчання — це не наповнення відра, а запалювання вогню.» — Вільям Батлер Єйтс",
    "«Успіх — це не випадковість, а результат важкої праці.» — Колін Пауелл"
]

# Відповіді міні-чат-бота
mini_chat_responses = {
    "привіт": "Привіт! Як настрій? □",
    "як справи?": "Усе чудово! А в тебе? □",
    "мені сумно": "Не засмучуйся! Все налагодиться. □",
    "хто ти?": "Я — твій асистент. Питай усе, що цікавить!",
    "що робиш?": "Чекаю твоїх команд. □",
    "дякую": "Завжди радий допомогти! □",
    "до побачення": "Бувай! Гарного дня! □"
}

# Функція для додавання повідомлення до логу
def add_to_log(message, log_panel, max_log_length, color):
    timestamp = datetime.now().strftime("[%H:%M:%S]")  # Створення мітки часу
    log_panel.content.controls.append(ft.Text(f"{timestamp} {message}", color=color))  # Додавання повідомлення до логу
    # Перевірка, чи не перевищує кількість повідомлень максимальний ліміт
    while len(log_panel.content.controls) > max_log_length:
        log_panel.content.controls.pop(0)  # Видалення найстарішого повідомлення
    log_panel.update()

# Функція для обробки команд і повідомлень
def process_command(command, log_panel, max_log_length):
    command = command.lower().strip()  # Перетворення команди в нижній регістр і видалення зайвих пробілів

    if not command:  # Якщо команда пуста
        add_to_log("□ Введіть команду або повідомлення!", log_panel, max_log_length, color="#FF4500")
        return

    # Словник стандартних команд
    commands = {
        "факт": "Факт: Людина може жити без їжі до 3 тижнів.",
        "допомога": "Команди: факт, допомога, час, дата, жарт, цитата, очистити.",
        "час": f"Поточний час: {datetime.now().strftime('%H:%M:%S')}",  # Показ поточного часу
        "дата": f"Сьогоднішня дата: {datetime.now().strftime('%d-%m-%Y')}",  # Показ поточної дати
        "жарт": random.choice(jokes),  # Вибір випадкового жарту
        "цитата": random.choice(quotes),  # Вибір випадкової цитати
        "очистити": "Лог очищено."  # Очищення логу
    }

    # Додавання команди користувача в лог
```

```
    add_to_log(f"Ви: {command}", log_panel, max_log_length, color="#FFFFFF")

    # Обробка команди "очистити"
    if command == "очистити":
        log_panel.content.controls.clear()  # Очищення всіх повідомлень у
логу
        log_panel.update()
        return

    # Перевірка стандартних команд
    if command in commands:
        response = commands[command]
        response_color = "#FFD700"
    # Перевірка міні-чат-бота
    elif command in mini_chat_responses:
        response = mini_chat_responses[command]
        response_color = "#90EE90"
    else:  # Якщо команда не знайдена
    response = "☐ Команда не розпізнана. Спробуйте ще раз або введіть 'допомога'."
        response_color = "#FF4500"

    # Виведення відповіді в лог
    add_to_log(f"Система: {response}", log_panel, max_log_length,
color=response_color)
```

Файл ui_components.py

```
import flet as ft
from db import load_user_data  # Імпортуємо функцію з db.py


# Функція для створення заголовка
def create_header():
    return ft.Container(
        content=ft.Text(
     "Асистент: Підвищення рівня □", # Текст заголовка
            size=30,  # Розмір шрифту
            color="#FFD700",  # Колір шрифту
            weight=ft.FontWeight.BOLD,  # Жирний шрифт
            text_align=ft.TextAlign.CENTER,  # Вирівнювання тексту по центру
        ),
        padding=20,  # Відступи всередині контейнера
        alignment=ft.alignment.center,  # Вирівнювання контейнера
    )


# Функція для створення статусу користувача
def create_user_status(user_id, navigate_to_user_details):
    updated_data = load_user_data(user_id)  # Завантажуємо дані користувача
    if not updated_data:
        updated_data = {
            "level": 1,
```

```
            "experience": 0
        }
    return ft.Container(
        content=ft.Column(
            [
                ft.Text(f"□ Рівень: {updated_data['level']}", size=18,
color="#FFD700", weight=ft.FontWeight.BOLD),
                ft.Text(f"□ Досвід: {updated_data['experience']}/100",
size=18, color="#FFFFFF"),
                ft.ElevatedButton("Деталі користувача", on_click=lambda _:
navigate_to_user_details()),
                # Кнопка для переходу до деталей користувача
            ],
            spacing=10,  # Простір між елементами
        ),
        padding=20,  # Відступи всередині контейнера
        margin=10,  # Зовнішні відступи
        bgcolor="#4A4A6A",  # Колір фону контейнера
        border_radius=15,  # Радіус скруглення контейнера
    )


# Функція для створення панелі логів
def create_log_panel():
    return ft.Container(
        content=ft.Column([], spacing=5),  # Панель для виведення логів
        padding=20,  # Відступи всередині контейнера
        margin=10,  # Зовнішні відступи
        bgcolor="#2E2E3E",  # Колір фону контейнера
        border_radius=15,  # Радіус скруглення
        expand=True,  # Розширення контейнера на весь доступний простір
    )


# Функція для створення поля вводу команд користувача
def create_user_input(on_submit):
    return ft.TextField(
        label="Введіть команду...('допомога')",  # Підказка для користувача
        label_style=ft.TextStyle(color="#B0B0B0"),  # Стиль підказки
        border_color="#FFD700",  # Колір межі
        color="#FFFFFF",  # Колір тексту
        cursor_color="#FFD700",  # Колір курсора
        on_submit=lambda e: on_submit(e.control.value),  # Обробник події при
натисканні Enter
    )
```

Файл education_page.py

```
import flet as ft
from db import get_all_subjects, get_topics_by_subject,
get_materials_by_topic, \
    get_quiz_by_topic, get_questions_and_answers, saves_quiz_result, \
    get_user_education, update_education_status
```

```
from pages.user_details_page import update_characteristics


def education_page(page, back_to_main, user_id):
    # Функция рендерит список всех предметов (разделов обучения)
    def render_subjects():
        subjects = get_all_subjects()  # Получаем список всех предметов из БД
        return ft.Row(
            [
                # Для каждого предмета создаем кнопку с названием,
                # при нажатии на которую показываются темы этого предмета
                ft.ElevatedButton(sub[1], on_click=lambda e, s=sub:
show_topics(s))
                for sub in subjects
            ],
            spacing=10,
            alignment=ft.MainAxisAlignment.CENTER
        )

    # Показывает список тем по выбранному предмету
    def show_topics(subject):
        subject_id, subject_name, _ = subject
        topics = get_topics_by_subject(subject_id)  # Получаем темы для
предмета
        user_education = get_user_education(user_id) or []  # Получаем
прогресс пользователя

        progress_by_topic = {}
        # Анализируем прогресс по каждой теме (сколько тестов пройдено)
        for row in user_education:
            _, uid, topic_id, test_passed, materials_unlocked, _, quiz_block
= row
            if topic_id not in progress_by_topic:
                progress_by_topic[topic_id] = {"passed_blocks": 0}
            if test_passed:
                progress_by_topic[topic_id]["passed_blocks"] += 1

        page.clean()  # Очищаем страницу
        page.add(
            ft.Column(
                [
                    ft.Text(f"Теми з {subject_name}", size=30,
color="#FFD700", weight=ft.FontWeight.BOLD),
                    ft.Divider(color="#FFD700", thickness=1),
                    # Кнопки с темами и прогрессом по ним
                    *[
                        ft.ElevatedButton(
                            f"{topic[2]} ({progress_by_topic.get(topic[0],
{}).get('passed_blocks', 0)}/3)",
                            on_click=lambda e, t=topic:
show_materials(subject, t)
                        )
                        for topic in topics
```

```
                    ],
                    ft.ElevatedButton("Назад", on_click=lambda _:
back_to_main())
                ],
                spacing=20,
                expand=True
            )
        )

    # Показывает учебные материалы и тесты для выбранной темы
    def show_materials(subject, topic):
        subject_id, subject_name, _ = subject
        topic_id, _, topic_name, topic_desc, level = topic

        all_materials = get_materials_by_topic(topic_id)  # Получаем
материалы темы
        quiz = get_quiz_by_topic(topic_id)  # Получаем тест для темы
        user_education = get_user_education(user_id) or []  # Прогресс
пользователя

        progress_by_topic = {}
        # Формируем словарь с информацией о пройденных тестах по блокам
        for row in user_education:
            _, uid, t_id, test_passed, materials_unlocked, _, quiz_block =
row
            if t_id not in progress_by_topic:
                progress_by_topic[t_id] = {}
            progress_by_topic[t_id][quiz_block] = test_passed

        topic_progress = progress_by_topic.get(topic_id, {})

        # Делим все материалы на 3 блока (для прохождения поэтапно)
        blocks = [all_materials[i::3] for i in range(3)]
        user_answers = {}  # Словарь для ответов пользователя на вопросы

        # Обработка выбора ответа пользователем
        def on_answer_selected(e, question_id):
            answer_id = int(e.control.value)
            user_answers[question_id] = answer_id

        # Обработка завершения теста для блока
        def submit_quiz(e, block_index):
            block_size = 1  # Размер блока (кол-во вопросов)
            start = block_index * block_size
            end = start + block_size
            all_questions = get_questions_and_answers(quiz[0])  # Получаем
все вопросы
            difficulty_order = {'початковий': 1, 'середній': 2, 'високий': 3}
            # Сортируем вопросы по уровню сложности
            all_questions = sorted(all_questions,
                                   key=lambda q:
difficulty_order.get(q.get('difficulty_level', 'початковий'), 1))
            block_questions = all_questions[start:end]
```

```
            score = 0
            # Подсчитываем количество правильных ответов
            for q in block_questions:
                correct = next((a for a in q["answers"] if a["is_correct"]),
None)
                if correct and user_answers.get(q["question_id"]) ==
correct["answer_id"]:
                    score += 1

            # Проверяем, прошел ли пользователь тест (70% и выше)
            passed = score >= (len(block_questions) * 0.7)
            saves_quiz_result(user_id, quiz[0], score, passed)  # Сохраняем
результат теста

            # Обновляем статус обучения в БД
            update_education_status(user_id, topic_id, test_passed=passed,
materials_unlocked=True, quiz_block=block_index)

            # Показываем результат и обновляем характеристики пользователя
            show_result(score, len(block_questions), passed, block_index)
            update_characteristics(user_id)

        # Запуск теста (показ вопросов и вариантов ответов)
        def start_quiz(e, block_index):
            page.clean()
            components = [
        ft.Text(f"Тест №{block_index + 1} по темі: {topic_name}", size=24, weight=ft.FontWeight.BOLD,
                        color="#FFD700")
            ]

            block_size = 1
            start = block_index * block_size
            end = start + block_size
            all_questions = get_questions_and_answers(quiz[0])
            difficulty_order = {'початковий': 1, 'середній': 2, 'високий': 3}
            all_questions = sorted(all_questions,
                                   key=lambda q:
difficulty_order.get(q.get('difficulty_level', 'початковий'), 1))
            block_questions = all_questions[start:end]

            if not block_questions:
                components.append(ft.Text("Питання для цього блоку не
знайдені.", color="red"))
                page.add(ft.Column(components))
                page.update()
                return

            # Отображаем вопросы и радиокнопки с ответами
            for q in block_questions:
                components.append(ft.Text(q["question_text"], size=18,
weight=ft.FontWeight.BOLD, color="#FFFFFF"))
                radio_buttons = [
```

```
                    ft.Radio(value=str(a["answer_id"]), label=a["text"],
label_style=ft.TextStyle(color="#FFFFFF"))
                    for a in q["answers"]
                ]
                radio_group = ft.RadioGroup(
                    value="",
                    content=ft.Column(radio_buttons),
                    on_change=lambda e, q_id=q["question_id"]:
on_answer_selected(e, q_id)
                )
                components.append(radio_group)

            # Кнопки для завершения теста и возврата к материалам
            components.append(ft.ElevatedButton("Завершити тест",
on_click=lambda e: submit_quiz(e, block_index)))
            components.append(ft.ElevatedButton("Назад", on_click=lambda _:
show_materials(subject, topic)))

            page.add(ft.Column(components, spacing=15, expand=True))
            page.update()

        # Показывает результат теста: баллы и статус прохождения
        def show_result(score, total, passed, block_index):
            page.clean()
            result_text = ft.Text(f"Тест завершено! Ваш результат: {score} з
{total}", size=20, color="#FFFFFF")
            status_text = ft.Text(
                "Ви успішно пройшли тест." if passed else "Ви не пройшли
тест.",
                color="#00FF00" if passed else "#FF0000"
            )
            components = [
                result_text,
                status_text,
                ft.ElevatedButton("Пройти ще раз", on_click=lambda e:
start_quiz(e, block_index)),
                ft.Divider(thickness=2),
                ft.ElevatedButton("Назад", on_click=lambda _:
show_materials(subject, topic))
            ]
            page.add(ft.Column(components, spacing=20,
alignment=ft.MainAxisAlignment.CENTER, expand=True))
            page.update()

        page.clean()
        components = [
            ft.Text(f"Навчальні матеріали по {topic_name}", size=30,
color="#FFD700", weight=ft.FontWeight.BOLD),
            ft.Text(topic_desc or "Опис відсутній.", size=16,
color="#FFFFFF"),
        ]

        # Перебираем 3 блока материалов и отображаем их, если предыдущий блок
пройден
```

```
        for i in range(3):
            prev_passed = i == 0 or topic_progress.get(i - 1) is True
            if prev_passed:
                block_materials = blocks[i]
                if block_materials:
                    components.append(ft.Text(f"Блок {i + 1}", size=20,
color="#FFA500", weight=ft.FontWeight.BOLD))
                    for mat in block_materials:
                        components.append(ft.Text(f"• {mat[2]}", size=16,
color="#FFFFFF"))
                    passed = topic_progress.get(i, False)
```

test_status = "☐ Пройдено" if passed else "☐ Не пройдено"

```
                    components.append(
                        ft.ElevatedButton(f"Тест до блоку {i + 1}
({test_status})", on_click=lambda e, i=i: start_quiz(e, i)))
                    components.append(ft.Divider(color="#555555"))
            else:
                # Блок заблокирован пока не пройден предыдущий
                components.append(ft.Text(f"Блок {i + 1} буде доступний після
проходження попереднього тесту.",
                                          size=16, color="#888888",
italic=True))

        components.append(ft.ElevatedButton("Назад", on_click=lambda _:
show_topics(subject)))
        page.add(ft.Column(components, spacing=15, expand=True))
        page.update()

    # Главная страница обучения: выбор предмета
    def back_to_main_page():
        page.clean()
        page.add(
            ft.Column(
                [
                    ft.Text("Виберіть предмет для навчання", size=30,
color="#FFD700", weight=ft.FontWeight.BOLD),
                    ft.Divider(color="#FFD700", thickness=1),
                    render_subjects(),  # Показываем кнопки с предметами
                    ft.ElevatedButton("Назад", on_click=lambda _:
back_to_main()),
                ],
                spacing=20,
                expand=True
            )
        )

    back_to_main_page()  # Показываем главную страницу с предметами при
запуске
```

Файл personal_development_page.py

```
# Імпортуємо необхідні бібліотеки
from datetime import datetime
```

```
import flet as ft
from db import get_connection, load_personal_goals, add_personal_goal,
load_user_data
import matplotlib.pyplot as plt
import matplotlib
import numpy as np
import io
import base64

# Кеш для збереження зображень радарних діаграм (щоб не будувати повторно)
radar_cache = {}


# Функція для створення радарної діаграми на основі характеристик користувача
def create_radar_chart(user_data: dict, norm: int = 80) -> str:
    matplotlib.use('Agg')  # Встановлюємо бекенд без GUI
    user_key = str(user_data)

    # Якщо зображення вже є в кеші — повертаємо його
    if user_key in radar_cache:
        return radar_cache[user_key]

    # Мітки для діаграми
    labels = ["Сила", "Швидкість", "Витривалість", "Баланс", "Інтелект"]
    values = [
        user_data["strength"],
        user_data["speed"],
        user_data["endurance"],
        user_data["balance"],
        user_data["intelligence"],
    ]
    norms = [norm] * len(values)

    # Розрахунок кутів для побудови діаграми
    angles = np.linspace(0, 2 * np.pi, len(labels), endpoint=False).tolist()
    values += values[:1]  # Замикання кола
    norms += norms[:1]
    angles += angles[:1]

    # Створення фігури та осі
    fig, ax = plt.subplots(figsize=(5, 5), subplot_kw=dict(polar=True))
    ax.plot(angles, norms, label="Норма", color="gray", linestyle="--")
    ax.plot(angles, values, label="Користувач", color="gold")
    ax.fill(angles, values, color="gold", alpha=0.4)

    # Налаштування зовнішнього вигляду діаграми
    ax.set_thetagrids(np.degrees(angles[:-1]), labels)
    ax.set_ylim(0, 100)
    ax.set_title("Характеристики")
    ax.legend(loc='upper right', bbox_to_anchor=(1.2, 1.1))

    # Збереження діаграми у форматі base64
```

```
    buffer = io.BytesIO()
    plt.savefig(buffer, format="png", bbox_inches="tight", dpi=150)
    buffer.seek(0)
    plt.close(fig)

    # Звільнення пам’яті
    del fig, ax

    # Кодуємо зображення та зберігаємо у кеш
    image_base64 = base64.b64encode(buffer.read()).decode("utf-8")
    radar_cache[user_key] = image_base64
    return image_base64


# Основна функція для сторінки особистого розвитку
def personal_development_page(page, back_to_main, user_id):
    page.clean()  # Очищення сторінки

    # Колонка для виведення списку цілей користувача
    goals_column = ft.Column(spacing=10)

    # Функція для оновлення списку цілей
    def update_goals_list():
        goals_column.controls.clear()
        goals = load_personal_goals(user_id)

        for goal in goals:
            goal_text = ft.Text(goal["goal"], color="#FFFFFF", size=16,
weight=ft.FontWeight.BOLD)
            if goal["is_completed"]:
                goal_text.style =
ft.TextStyle(decoration=ft.TextDecoration.LINE_THROUGH, color="#00FF00")

            goals_column.controls.append(
                ft.Row(
                    [
                        goal_text,
                        # Чекбокс для позначення завершення цілі
                        ft.Checkbox(
                            value=goal["is_completed"],
                            on_change=lambda e, gid=goal["id"], status=not
goal["is_completed"]: toggle_goal_completed(
                                gid, status) or update_goals_list()
                        ),
                        # Кнопка для видалення цілі
                        ft.IconButton(ft.icons.DELETE,
                                      on_click=lambda e, gid=goal["id"]:
delete_goal(gid) or update_goals_list()),
                    ],
                    alignment=ft.MainAxisAlignment.START,
                    spacing=10,
                )
            )
```

```
        page.update()

    # Функція для створення нової цілі
    def set_personal_goal(e):
        goal_input = ft.TextField(label="Введіть вашу ціль", width=300)

        def save_goal(event):
            goal_text = goal_input.value.strip()
            if goal_text:
                add_personal_goal(user_id, goal_text)
                update_goals_list()
                dialog.open = False
                page.update()

        dialog = ft.AlertDialog(
            title=ft.Text("Нова ціль", color="#FFD700"),
            content=goal_input,
            actions=[
                ft.TextButton("Скасувати", on_click=lambda e:
close_dialog()),
                ft.TextButton("Зберегти", on_click=save_goal),
            ],
        )

        page.dialog = dialog
        dialog.open = True
        page.update()

    # Функція для закриття діалогового вікна
    def close_dialog():
        page.dialog.open = False
        page.update()

    # Функція для видалення цілі
    def delete_goal(goal_id):
        with get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute("DELETE FROM personal_goals WHERE id = %s;",
(goal_id,))
                conn.commit()

    # Функція для оновлення статусу цілі (виконано/не виконано)
    def toggle_goal_completed(goal_id, new_status):
        with get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute(
                    "UPDATE personal_goals SET is_completed = %s, updated_at
= %s WHERE id = %s;",
                    (new_status, datetime.now(), goal_id)
                )
                conn.commit()

    # Блок з кнопками вибору напрямків розвитку
```

```
    development_options = ft.Column(
        [
            ft.ElevatedButton("Фізичний розвиток", on_click=lambda _:
choose_physical_development()),
            ft.ElevatedButton("Інтелектуальний розвиток", on_click=lambda _:
choose_intellectual_development()),
            ft.ElevatedButton("Емоційне здоров'я", on_click=lambda _:
choose_emotional_health()),
            ft.ElevatedButton("Встановити ціль", on_click=set_personal_goal),
        ],
        spacing=10,
    )

    # Функції для переходу до різних напрямків розвитку
    def choose_physical_development():
        page.clean()
        page.add(ft.Text("Фізичний розвиток", size=24, color="#FFD700"))
        page.add(ft.Text("Рекомендації: заняття спортом, правильне
харчування, режим сну.", color="#FFFFFF"))
        page.add(ft.ElevatedButton("Назад", on_click=lambda _:
personal_development_page(page, back_to_main, user_id)))

    def choose_intellectual_development():
        page.clean()
        page.add(ft.Text("Інтелектуальний розвиток", size=24,
color="#FFD700"))
        page.add(ft.Text("Рекомендації: читання книг, онлайн-курси, вивчення
мов.", color="#FFFFFF"))
        page.add(ft.ElevatedButton("Назад", on_click=lambda _:
personal_development_page(page, back_to_main, user_id)))

    def choose_emotional_health():
        page.clean()
        page.add(ft.Text("Емоційне здоров'я", size=24, color="#FFD700"))
        page.add(ft.Text("Рекомендації: медитація, спілкування з близькими,
психологічна підтримка.", color="#FFFFFF"))
        page.add(ft.ElevatedButton("Назад", on_click=lambda _:
personal_development_page(page, back_to_main, user_id)))

    # Завантаження характеристик користувача та створення радарної діаграми
    user_data = load_user_data(user_id)
    radar_base64 = create_radar_chart(user_data)
    radar_image = ft.Image(src_base64=radar_base64, width=350, height=350)

    # Головний блок із діаграмою та кнопками розвитку
    blog = ft.Row([development_options,
                   ft.Text("Ваші характеристики:", size=20, color="#FFD700",
weight=ft.FontWeight.BOLD),
                   radar_image
                   ], spacing=50)

    # Основна структура сторінки
    page.add(
        ft.Column(
```

```
            [
                ft.Text("Особистий розвиток", size=30, color="#FFD700",
weight=ft.FontWeight.BOLD),
                ft.Text("Тут можна вибрати напрямок для розвитку своїх
особистих якостей.", color="#FFFFFF"),
                blog,
                ft.Divider(color="#FFD700", thickness=1),
                ft.Text("Ваші персональні цілі:", size=20, color="#FFD700",
weight=ft.FontWeight.BOLD),
                goals_column,
                ft.ElevatedButton("Повернутися", on_click=lambda _:
back_to_main()),
            ],
            spacing=20,
            expand=True,
        )
    )

    # Перший виклик для відображення цілей
    update_goals_list()
```

Файл sport_page.py

```
import flet as ft
import random
import functools
from db import get_categories_with_exercises, get_user_count,
update_user_count
from pages.user_details_page import update_characteristics

# Отримуємо словник категорій з відповідними вправами з бази даних
categories = get_categories_with_exercises()

# Основна функція, яка створює сторінку зі спортивними завданнями
def sport_page(page: ft.Page, back_to_main, current_user_id):
    # Створюємо чекбокси для кожної категорії вправ
    category_checkboxes = [
        ft.Checkbox(
            label=category,
            value=False,
            label_style=ft.TextStyle(color="#C0C0C0"),
            on_change=lambda e, category=category:
update_category_selection(category),
        )
        for category in categories.keys()
    ]

    # Колонка для відображення згенерованих завдань
    task_output = ft.Column()
    # Колонка для відображення виконаних завдань
    completed_tasks = ft.Column()

    # Функція повертає список вибраних категорій
```

```
    def selected_categories():
        return [cb.label for cb in category_checkboxes if cb.value]

    # Допоміжна функція — оновлює інформацію про вибрану категорію (для
тестування/відладки)
    def update_category_selection(category):
        task_output.controls = [ft.Text(f"Вибрана категорія: {category}",
color="#C0C0C0")]
        task_output.update()

    # Функція генерує до 3 завдань з випадкових вибраних категорій
    def generate_tasks(e):
        task_output.controls.clear()
        completed_tasks.controls.clear()
        completed_tasks.update()

        # Якщо не обрано жодної категорії — показати попередження
        if not selected_categories():
            task_output.controls.append(
                ft.Text("Будь ласка, виберіть хоча б одну категорію!",
color="red")
            )
            task_output.update()
            return

        selected_exercises = set()
        available_exercises = []

        # Формуємо список доступних вправ із вибраних категорій
        for category in selected_categories():
            for exercise_name, default_count in categories[category]:
                # Отримуємо індивідуальну кількість повторень для вправи
                count = get_user_count(current_user_id, category,
exercise_name) or default_count
                available_exercises.append((category, exercise_name, count))

        # Перемішуємо список вправ
        random.shuffle(available_exercises)

        # Додаємо до 3 унікальних вправ
        while len(selected_exercises) < 3 and available_exercises:
            chosen_category, task_name, count = available_exercises.pop()
            if task_name not in selected_exercises:
                selected_exercises.add(task_name)

                # Створюємо елемент зі вправою та кнопками для оцінки
складності
                task_row = ft.Row([
                    ft.Text(f"{task_name} ({chosen_category}) - {count}
разів", size=16, color="#D3D3D3"),
                    ft.ElevatedButton("Легко",
on_click=functools.partial(rate_task, task_name, "Легко")),
                    ft.ElevatedButton("Нормально",
on_click=functools.partial(rate_task, task_name, "Нормально")),
```

```
                    ft.ElevatedButton("Складно",
on_click=functools.partial(rate_task, task_name, "Складно")),
                ])
                task_output.controls.append(task_row)

        task_output.update()
        # Оновлюємо характеристики користувача після генерації завдань
        update_characteristics(current_user_id)

    # Функція обробляє оцінку складності вправи користувачем
    def rate_task(task_name, rating, e):
        # Знаходимо категорію для цієї вправи
        category = next(
            (cat for cat, exercises in categories.items() if any(name ==
task_name for name, _ in exercises)),
            None
        )
        if not category:
            return

        # Отримуємо поточну кількість повторень для вправи
        current_count = get_user_count(current_user_id, category, task_name)
        if current_count is None:
            current_count = next((count for name, count in
categories[category] if name == task_name), 5)

        # Змінюємо складність відповідно до оцінки
        if rating == "Складно":
            current_count = max(5, current_count - 5)  # Мінімум 5 повторень
        elif rating == "Легко":
            current_count += 5
        # Якщо "Нормально", не змінюємо count

        # Оновлюємо кількість у базі даних
        update_user_count(current_user_id, category, task_name,
current_count)

        # Додаємо вправу до списку виконаних
        completed_tasks.controls.append(
            ft.Text(f"Виконано: {task_name} - {rating} ({current_count}
разів)", color="green")
        )

        # Видаляємо вправу зі списку активних завдань
        for row in task_output.controls[:]:
            if any(task_name in str(c) for c in row.controls):
                task_output.controls.remove(row)

        task_output.update()
        completed_tasks.update()

    # Очищаємо сторінку та додаємо всі елементи
    page.clean()
```

```
    page.add(
        ft.Column([
            ft.Text("□♂□ Спорт", size=30, color="#FFA500",
weight=ft.FontWeight.BOLD),  # Заголовок
            *[ft.Row(category_checkboxes[i:i + 3]) for i in range(0,
len(category_checkboxes), 3)],  # Рядки з чекбоксами
            ft.ElevatedButton("Згенерувати завдання",
on_click=generate_tasks),  # Кнопка генерації
            task_output,  # Виведення завдань
            ft.Text("□ Виконані завдання:", size=18, color="#C0C0C0", weight=ft.FontWeight.BOLD),  #
Заголовок для виконаних
            completed_tasks,  # Виведення виконаних
            ft.ElevatedButton("Повернутися", on_click=lambda _:
back_to_main()),  # Кнопка повернення
        ], spacing=20)
    )
```

Файл user_details_page.py

```
import flet as ft
from db import load_user_data, save_user_data, get_user_progress,
get_user_education, get_categories_with_exercises, \
    get_connection, get_category_to_stat_map


# □ Функція для оновлення характеристик та досвіду користувача на основі
виконаних вправ та освітнього прогресу
def update_characteristics(user_id):
    user_data = load_user_data(user_id)

    # Початкові значення змін характеристик
    total_strength = 0
    total_speed = 0
    total_endurance = 0
    total_balance = 0
    total_intelligence = 0

    # Збереження попередніх значень характеристик
    old_strength = user_data["strength"]
    old_speed = user_data["speed"]
    old_endurance = user_data["endurance"]
    old_balance = user_data["balance"]
    old_intelligence = user_data["intelligence"]

    # Отримуємо категорії вправ та карту відповідності категорій
характеристикам
    categories = get_categories_with_exercises()
    category_to_progress_type = get_category_to_stat_map()

    conn = get_connection()
    cursor = conn.cursor()

    # □ Перебираємо категорії та вправи, щоб розрахувати прогрес
```

```
    for category, exercises in categories.items():
        for exercise_name, base_count in exercises:
            user_count, last_count = get_user_progress(user_id, category,
exercise_name)

            delta = user_count - last_count
            if delta > 0:
                progress_type = category_to_progress_type.get(category)

                # Додаємо приріст до відповідної характеристики
                if progress_type == "strength":
                    total_strength += delta
                elif progress_type == "speed":
                    total_speed += delta
                elif progress_type == "endurance":
                    total_endurance += delta
                elif progress_type == "intelligence":
                    total_intelligence += delta
                elif progress_type == "balance":
                    total_balance += delta

                # Оновлюємо last_count у базі даних
                cursor.execute("""
                        UPDATE user_progress
                        SET last_count = %s
                        WHERE user_id = %s AND category = %s AND
exercise_name = %s
                    """, (user_count, user_id, category, exercise_name))

    # □ Додаткове оновлення інтелекту за завершені освітні блоки, які ще не
були враховані
    cursor.execute("""
           SELECT id, quiz_block
           FROM user_education_progress
           WHERE user_id = %s AND test_passed = TRUE AND progress_counted =
FALSE
       """, (user_id,))
    new_intelligence_rows = cursor.fetchall()

    for row_id, quiz_block in new_intelligence_rows:
        if quiz_block == 0:
            total_intelligence += 5
        elif quiz_block == 1:
            total_intelligence += 8
        elif quiz_block == 2:
            total_intelligence += 10

        # Позначаємо, що цей блок вже врахований
        cursor.execute("""
               UPDATE user_education_progress
               SET progress_counted = TRUE
               WHERE id = %s
           """, (row_id,))
```

```
    # ☐ Оновлюємо характеристики користувача з урахуванням обмеження максимуму 100
    user_data["strength"] = min(user_data["strength"] + total_strength * 0.2,
100)
    user_data["speed"] = min(user_data["speed"] + total_speed * 0.2, 100)
    user_data["endurance"] = min(user_data["endurance"] + total_endurance *
0.2, 100)
    user_data["balance"] = min(user_data["balance"] + total_endurance * 0.2,
100)
    user_data["intelligence"] = min(user_data["intelligence"] +
total_intelligence * 0.2, 100)

    # □ Перевірка змін характеристик
    characteristics_increased = (
            user_data["strength"] > old_strength or
            user_data["speed"] > old_speed or
            user_data["endurance"] > old_endurance or
            user_data["balance"] > old_balance or
            user_data["intelligence"] > old_intelligence
    )

    characteristics_decreased = (
            user_data["strength"] < old_strength or
            user_data["speed"] < old_speed or
            user_data["endurance"] < old_endurance or
            user_data["balance"] < old_balance or
            user_data["intelligence"] < old_intelligence
    )

    # □ Оновлюємо досвід в залежності від змін
    if characteristics_increased:
        user_data["experience"] = min(user_data["experience"] + 10, 200)
    elif characteristics_decreased:
        user_data["experience"] = max(user_data["experience"] - 10, 0)

    # □ Підвищення рівня при досягненні 100 досвіду
    while user_data["experience"] >= 100:
        user_data["experience"] -= 100
        user_data["level"] += 1

    conn.commit()
    conn.close()

    # □ Збереження оновлених даних користувача
    save_user_data(user_id, user_data)


# □ Сторінка з деталями користувача
def user_details_page(page, back_to_main, user_data):
    # □ Відображення характеристик користувача у вигляді трьох колонок
    def render_stats_section():
        stats_section = ft.Row(
            [
```

```
                ft.Column(
                    [
                        ft.Text(f"Сила: {user_data['strength']}", size=16,
color="#FFFFFF"),
                        ft.Text(f"Координація: {user_data['balance']}",
size=16, color="#FFFFFF"),
                    ],
                    alignment=ft.MainAxisAlignment.CENTER,
                    spacing=10,
                ),
                ft.Column(
                    [
                        ft.Text(f"Інтелект: {user_data['intelligence']}",
size=16, color="#FFFFFF")
                    ],
                    alignment=ft.MainAxisAlignment.CENTER,
                    spacing=10,
                ),
                ft.Column(
                    [
                        ft.Text(f"Швидкість: {user_data['speed']}", size=16,
color="#FFFFFF"),
                        ft.Text(f"Витривалість: {user_data['endurance']}",
size=16, color="#FFFFFF"),
                    ],
                    alignment=ft.MainAxisAlignment.CENTER,
                    spacing=10,
                ),
            ],
            alignment=ft.MainAxisAlignment.CENTER,
        )
        return stats_section

    # □ Відображення рівня та досвіду користувача
    def render_level_section():
        current_experience = user_data["experience"]
        max_experience = 100
        experience_progress = current_experience / max_experience

        return ft.Column(
            [
                ft.Text(f"□ Рівень: {user_data['level']}", size=20,
color="#FFD700", weight=ft.FontWeight.BOLD),
                ft.Container(
                    content=ft.ProgressBar(value=experience_progress,
color="#FFD700"),
                    height=20,
                    border_radius=5,
                    bgcolor="#3E3E56",
                    expand=True,
                ),
                ft.Container(
                    content=ft.Text(
```

```
                        f"{current_experience}/{max_experience}",
                        size=14,
                        color="#FFFFFF",
                        text_align=ft.TextAlign.RIGHT,
                    ),
                    alignment=ft.alignment.bottom_right,
                    padding=ft.Padding(0, 2, 0, 0),
                    expand=True,
                ),
            ],
            spacing=10,
        )

    # 🔹 Поля для редагування імені та віку
    name_input = ft.TextField(
        label="Ім'я",
        label_style=ft.TextStyle(color="#B0B0B0"),
        value=user_data["name"],
        color="#FFFFFF",
        border_color="#FFD700"
    )
    age_input = ft.TextField(
        label="Вік",
        label_style=ft.TextStyle(color="#B0B0B0"),
        value=user_data["age"],
        color="#FFFFFF",
        border_color="#FFD700"
    )

    # 🔹 Обробка кнопки повернення — зберігаємо зміни та переходимо назад
    def on_back():
        user_data["name"] = name_input.value
        user_data["age"] = age_input.value
        save_user_data(user_data["id"], user_data)
        back_to_main()

    # 🔹 Оновлення сторінки з актуальними даними користувача
    def update_page_content():
        updated_data = load_user_data(user_data["id"])
        user_data.update(updated_data)  # синхронізація

        update_characteristics(user_data["id"])  # оновлення характеристик

        updated_data = load_user_data(user_data["id"])
        user_data.update(updated_data)

        page.clean()
        page.add(
            ft.Column(
                [
                    ft.Text("Деталі користувача", size=30, color="#FFD700",
weight=ft.FontWeight.BOLD),
```

```
                    name_input,
                    age_input,
                    render_level_section(),
                    ft.Divider(color="#FFD700", thickness=1),
                    render_stats_section(),
                    ft.Divider(color="#FFD700", thickness=1),
                    ft.ElevatedButton("Повернутися", on_click=lambda _:
on_back()),
                ],
                spacing=20,
                expand=True,
            )
        )

    # □ Ініціалізація сторінки
    update_page_content()
```

Файл db.py

```
import psycopg2
from psycopg2.extras import RealDictCursor

# Налаштування підключення до БД
db_config = {
    "host": "localhost",
    "port": "5432",
    "database": "diploma_app",
    "user": "postgres",
    "password": "turtur"
}


# Функція для створення нового підключення до БД
def get_connection():
    return psycopg2.connect(**db_config)


# Тимчасова функція для хешування пароля (поки що просто повертає пароль без
змін)
def hash_password(password): return password  # тимчасово


# Функція перевірки пароля (порівнює простий текст пароля)
def verify_password(raw_password, stored_password): return raw_password ==
stored_password


# Функція створення нового користувача в базі даних
def create_user(username, raw_password, role_id=1):
    conn = get_connection()
    cursor = conn.cursor()
    try:
        hashed = hash_password(raw_password)  # хешуємо пароль
```

```
        cursor.execute("""
            INSERT INTO users (name, password_hash, role_id)
            VALUES (%s, %s, %s)
            RETURNING id;
        """, (username, hashed, role_id))
        user_id = cursor.fetchone()[0]  # отримуємо ID створеного користувача
        conn.commit()
        return user_id
    finally:
        cursor.close()
        conn.close()


# Функція отримання користувача за ім'ям (username)
def get_user_by_username(username):
    conn = get_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("""
            SELECT users.id, users.name, users.password_hash, roles.name as
role_name
            FROM users
            JOIN roles ON users.role_id = roles.id
            WHERE users.name = %s;
        """, (username,))
        row = cursor.fetchone()
        if row:
            return {
                "id": row[0],
                "username": row[1],
                "password_hash": row[2],
                "role": row[3]
            }
        return None
    finally:
        cursor.close()
        conn.close()


# --- Функції роботи з даними користувача ---
# Завантажуємо дані користувача за його ID
def load_user_data(user_id):
    conn = get_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("""
            SELECT id, name, age, strength, intelligence, speed, endurance,
balance, level, experience
            FROM users
            WHERE id = %s;
        """, (user_id,))
        row = cursor.fetchone()
        if row:
```

```
            return {
                "id": user_id,  # ID користувача
                "name": row[1],
                "age": row[2],
                "strength": row[3],
                "intelligence": row[4],
                "speed": row[5],
                "endurance": row[6],
                "balance": row[7],
                "level": row[8],
                "experience": row[9]
            }
        return None
    finally:
        cursor.close()
        conn.close()


# Збереження оновлених даних користувача в базу
def save_user_data(user_id, user_data):
    conn = get_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("""
            UPDATE users SET
                name = %s,
                age = %s,
                strength = %s,
                intelligence = %s,
                speed = %s,
                endurance = %s,
                balance = %s,
                level = %s,
                experience = %s,
                updated_at = CURRENT_TIMESTAMP
            WHERE id = %s;
        """, (
            user_data["name"],
            user_data["age"],
            int(user_data["strength"]),
            int(user_data["intelligence"]),
            int(user_data["speed"]),
            int(user_data["endurance"]),
            int(user_data["balance"]),
            int(user_data["level"]),
            int(user_data["experience"]),
            user_id
        ))
        conn.commit()
    finally:
        cursor.close()
        conn.close()
```

```
# --- Функції, пов'язані з ролями користувачів ---
# Отримати роль користувача за його ID
def get_user_role(user_id):
    conn = get_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("""
            SELECT roles.name
            FROM users
            JOIN roles ON users.role_id = roles.id
            WHERE users.id = %s;
        """, (user_id,))
        row = cursor.fetchone()
        return row[0] if row else None
    finally:
        cursor.close()
        conn.close()

# Ініціалізація таблиці прогресу користувача, якщо її ще немає
def initialize_user_progress_table():
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS user_progress (
            category TEXT,
            exercise TEXT,
            count INTEGER,
            PRIMARY KEY (category, exercise)
        )
    """)
    conn.commit()
    conn.close()

# Оновлення прогресу користувача (кількість виконань вправи)
def update_user_progress(user_id, category, exercise, count):
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("""
        INSERT INTO user_progress (user_id, category, exercise, count)
        VALUES (%s, %s, %s, %s)
        ON CONFLICT (user_id, category, exercise)
        DO UPDATE SET count = EXCLUDED.count
    """, (user_id, category, exercise, count))
    conn.commit()
    conn.close()

# Отримати прогрес користувача за категорією та вправою
def get_user_progress(user_id, category, exercise_name):
    conn = get_connection()
    cursor = conn.cursor()
```

```
    cursor.execute("""
        SELECT count, last_count FROM user_progress
        WHERE user_id = %s AND category = %s AND exercise_name = %s
    """, (user_id, category, exercise_name))
    result = cursor.fetchone()
    conn.close()
    return result if result else (0, 0)

# Оновити кількість виконань вправи користувачем
def update_user_count(user_id, category, exercise_name, count):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute("""
        INSERT INTO user_progress (user_id, category, exercise_name, count,
updated_at)
        VALUES (%s, %s, %s, %s, CURRENT_TIMESTAMP)
        ON CONFLICT (user_id, category, exercise_name)
        DO UPDATE SET count = EXCLUDED.count, updated_at = CURRENT_TIMESTAMP
    """, (user_id, category, exercise_name, count))
    conn.commit()
    cur.close()
    conn.close()

# Отримати кількість виконань вправи користувачем
def get_user_count(user_id, category, exercise_name):
    conn = get_connection()
    cur = conn.cursor()
    cur.execute("""
        SELECT count FROM user_progress
        WHERE user_id = %s AND category = %s AND exercise_name = %s
    """, (user_id, category, exercise_name))
    result = cur.fetchone()
    cur.close()
    conn.close()
    return result[0] if result else None

# Отримати всі категорії вправ з їхніми вправами (список)
def get_categories_with_exercises():
    conn = get_connection()
    cursor = conn.cursor()
    query = """
        SELECT ec.name AS category_name, et.name AS exercise_name
        FROM exercise_categories ec
        LEFT JOIN exercise_templates et ON et.category_id = ec.id
        ORDER BY ec.name
    """
    cursor.execute(query)
    rows = cursor.fetchall()
    conn.close()

    categories = {}
    for category_name, exercise_name in rows:
```

```
        if category_name not in categories:
            categories[category_name] = []
        if exercise_name:
            categories[category_name].append((exercise_name, 5))  # 5 -
кількість за замовчуванням
    return categories

# Отримати відповідність категорій вправ до статів користувача
def get_category_to_stat_map():
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT name, related_stat FROM exercise_categories")
    rows = cursor.fetchall()
    conn.close()
    return {name: stat for name, stat in rows}

# Збереження результату тестування користувача
def save_quiz_result(user_id, quiz_id, score, passed):
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("""
        INSERT INTO quiz_results (user_id, quiz_id, score, passed)
        VALUES (%s, %s, %s, %s)
        ON CONFLICT (user_id, quiz_id)
        DO UPDATE SET score = EXCLUDED.score, passed = EXCLUDED.passed
    """, (user_id, quiz_id, score, passed))
    conn.commit()
    cursor.close()
    conn.close()


# --- Функції для роботи з предметами, темами, матеріалами та тестами ---
# 1. Отримати всі предмети
def get_all_subjects():
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM subjects")
    return cursor.fetchall()


# 2. Отримати теми за subject_id
def get_topics_by_subject(subject_id):
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM topics WHERE subject_id = %s",
(subject_id,))
    return cursor.fetchall()


# 3. Отримати матеріали за topic_id
def get_materials_by_topic(topic_id):
    conn = get_connection()
    cursor = conn.cursor()
```

```
    cursor.execute("SELECT * FROM materials WHERE topic_id = %s",
(topic_id,))
    return cursor.fetchall()


# 4. Отримати тест за topic_id
def get_quiz_by_topic(topic_id):
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM quizzes WHERE topic_id = %s", (topic_id,))
    return cursor.fetchone()


# 5. Отримати питання і відповіді для тесту
def get_questions_and_answers(quiz_id):
    conn = get_connection()
    cursor = conn.cursor()
    # Отримати питання з урахуванням рівня складності
    cursor.execute("""
        SELECT id, text, difficulty_level
        FROM questions
        WHERE quiz_id = %s
        ORDER BY difficulty_level
    """, (quiz_id,))
    questions = cursor.fetchall()
    result = []
    for question in questions:
        question_id, question_text, difficulty_level = question
        # Отримати відповіді для кожного питання
        cursor.execute("""
            SELECT id, text, is_correct
            FROM answers
            WHERE question_id = %s
        """, (question_id,))
        answers = cursor.fetchall()
        result.append({
            "question_id": question_id,
            "question_text": question_text,
            "difficulty_level": difficulty_level,
            "answers": [
                {"answer_id": a[0], "text": a[1], "is_correct": a[2]} for a
in answers
            ]
        })
    return result


# 6. Зберегти результат тесту
def saves_quiz_result(user_id, quiz_id, score, passed):
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("""
        INSERT INTO quiz_results (user_id, quiz_id, score, passed)
```

```
        VALUES (%s, %s, %s, %s)
    """, (user_id, quiz_id, score, passed))
    conn.commit()


# 7. Отримати прогрес користувача
def get_user_education(user_id):
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM user_education_progress WHERE user_id =
%s", (user_id,))
    return cursor.fetchall()


# 8. Оновити або додати запис у таблицю education
def update_education_status(user_id, topic_id, test_passed=False,
materials_unlocked=False, quiz_block=0):
    conn = get_connection()
    cursor = conn.cursor()
    # Перевірка, чи існує запис
    cursor.execute("""
        SELECT * FROM user_education_progress
        WHERE user_id = %s AND topic_id = %s AND quiz_block = %s
    """, (user_id, topic_id, quiz_block))
    existing = cursor.fetchone()
    if existing:
        # Оновити наявний запис (не чіпаємо progress_counted!)
        cursor.execute("""
            UPDATE user_education_progress
            SET test_passed = %s, materials_unlocked = %s
            WHERE user_id = %s AND topic_id = %s
        """, (test_passed, materials_unlocked, existing[0]))
    else:
        # Додати новий запис — і тут progress_counted автоматично = FALSE за
замовчуванням
        cursor.execute("""
            INSERT INTO user_education_progress (user_id, topic_id,
test_passed, materials_unlocked, quiz_block)
            VALUES (%s, %s, %s, %s, %s)
        """, (user_id, topic_id, test_passed, materials_unlocked,
quiz_block))
    conn.commit()
    conn.close()


def load_personal_goals(user_id):
    with get_connection() as conn:
        with conn.cursor(cursor_factory=RealDictCursor) as cur:
            cur.execute("SELECT id, goal, is_completed FROM personal_goals
WHERE user_id = %s ORDER BY id;", (user_id,))
            return cur.fetchall()
```

```
def add_personal_goal(user_id, goal_text):
    with get_connection() as conn:
        with conn.cursor() as cur:
            cur.execute(
                "INSERT INTO personal_goals (user_id, goal) VALUES (%s,
%s);",
                (user_id, goal_text)
            )
            conn.commit()


def delete_user(user_id):
    conn = get_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("DELETE FROM users WHERE id = %s", (user_id,))
        conn.commit()
        return cursor.rowcount > 0  # True якщо користувача видалено
    finally:
        cursor.close()
        conn.close()


def is_admin(user_id):
    conn = get_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("SELECT role_id FROM users WHERE id = %s;",
(user_id,))
        row = cursor.fetchone()
        if row:
            return row[0] == 2  # 2 — це адміністратор
        return False
    finally:
        cursor.close()
        conn.close()


def load_all_users():
    conn = get_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("""
            SELECT id, name, age, strength, intelligence, speed, endurance,
balance, level, experience, role_id
            FROM users
            WHERE role_id = 1;
        """)
        rows = cursor.fetchall()
        users = []
        for row in rows:
            users.append({
                "id": row[0],
```

```
                "name": row[1],
                "age": row[2],
                "strength": row[3],
                "intelligence": row[4],
                "speed": row[5],
                "endurance": row[6],
                "balance": row[7],
                "level": row[8],
                "experience": row[9],
                "role_id": row[10]
            })
        return users
    finally:
        cursor.close()
        conn.close()
```

Файл admin.py

```
import flet as ft
from db import load_all_users, delete_user  # Імпорт функцій для роботи з базою: завантаження та видалення користувачів


def admin_page(page: ft.Page):
    users_data = load_all_users()  # Завантажуємо усіх користувачів з бази даних
    user_controls = []  # Список елементів інтерфейсу для відображення користувачів

    # Функція оновлення списку користувачів, з можливістю сортування за типом
    def refresh_user_list(filter_type=None):
        user_controls.clear()  # Очищуємо поточний список елементів

        # Якщо вибрано сортування за ім'ям — сортуємо, інакше залишаємо порядок як є
        filtered_users = sorted(users_data, key=lambda x: x['name']) if filter_type == "name" else users_data

        # Обмеження відображення — максимум 3 користувачі
        limited_users = filtered_users[:3]

        for user in limited_users:
            user_controls.append(
                ft.Container(
                    content=ft.Column([

                        # Рядок із іменем користувача та кнопкою "Видалити"
                        ft.Row([
                            ft.Icon(name=ft.icons.PERSON, color=ft.colors.WHITE),
                            ft.Text(f"Ім'я: {user['name']}", size=18, weight=ft.FontWeight.BOLD, color=ft.colors.WHITE),
                            ft.ElevatedButton("Видалити", color=ft.colors.BLUE,
```

```
                                                on_click=lambda e,
uid=user['id']: remove_user(uid))  # Виклик функції видалення з id
користувача
                        ], alignment=ft.MainAxisAlignment.SPACE_BETWEEN),

                        # Рядок із рівнем і досвідом користувача
                        ft.Row([
                            ft.Icon(name=ft.icons.LEADERBOARD,
color=ft.colors.WHITE),
                            ft.Text(f"Рівень: {user.get('level', 0)}",
color=ft.colors.WHITE),
                            ft.Icon(name=ft.icons.STAR,
color=ft.colors.WHITE),
                            ft.Text(f"Досвід: {user.get('experience', 0)}",
color=ft.colors.WHITE),
                        ], spacing=10),

                        # Рядок із характеристиками користувача (сила,
інтелект, швидкість, витривалість, баланс)
                        ft.Row([
                            ft.Icon(name=ft.icons.FITNESS_CENTER,
color=ft.colors.WHITE),
                            ft.Text(f"Сила: {user.get('strength', 0)}",
color=ft.colors.WHITE),
                            ft.Icon(name=ft.icons.PSYCHOLOGY,
color=ft.colors.WHITE),
                            ft.Text(f"Інтелект: {user.get('intelligence',
0)}", color=ft.colors.WHITE),
                            ft.Icon(name=ft.icons.SPEED,
color=ft.colors.WHITE),
                            ft.Text(f"Швидкість: {user.get('speed', 0)}",
color=ft.colors.WHITE),
                            ft.Icon(name=ft.icons.FAVORITE,
color=ft.colors.WHITE),
                            ft.Text(f"Витривалість: {user.get('endurance',
0)}", color=ft.colors.WHITE),
                            ft.Icon(name=ft.icons.ACCESSIBILITY,
color=ft.colors.WHITE),
                            ft.Text(f"Баланс: {user.get('balance', 0)}",
color=ft.colors.WHITE),
                        ], spacing=10),

                    ]),
                    padding=10,
                    bgcolor=ft.colors.with_opacity(0.08,
ft.colors.BLUE_GREY),  # Фон контейнера з прозорістю
                    border_radius=10  # Закруглення кутів контейнера
                )
            )

        # Оновлення колонки з користувачами на сторінці
        users_column.controls = user_controls
        page.update()  # Оновлення сторінки для відображення змін

    # Функція видалення користувача за ID
```

```
    def remove_user(uid):
        delete_user(uid)  # Видаляємо користувача з бази даних
        page.snack_bar = ft.SnackBar(ft.Text("Користувача видалено"))  #
Показуємо сповіщення про успішне видалення
        page.snack_bar.open = True

        # Оновлюємо локальний список користувачів, виключаючи видаленого
        new_list = [u for u in users_data if u['id'] != uid]
        users_data.clear()
        users_data.extend(new_list)
        refresh_user_list()  # Оновлюємо список на сторінці

    users_column = ft.Column()  # Створюємо колонку для виводу користувачів

    page.clean()  # Очищаємо сторінку перед додаванням нового вмісту
    page.add(
        ft.Column([
            ft.Text("□ Панель адміністратора", size=24,
weight=ft.FontWeight.BOLD, color=ft.colors.WHITE),

            # Рядок з кнопками сортування
            ft.Row([
                ft.Text("Сортувати:", color=ft.colors.WHITE),
                ft.ElevatedButton("□ За ім’ям", color=ft.colors.BLUE,
on_click=lambda e: refresh_user_list("name")),
                ft.ElevatedButton("□ За рівнем", color=ft.colors.BLUE,
on_click=lambda e: refresh_user_list("level")),
            ]),

            ft.Divider(color=ft.colors.WHITE),
            users_column,  # Колонка з користувачами, сюди додаються елементи
            ft.Divider(color=ft.colors.WHITE),
        ],
            scroll=ft.ScrollMode.AUTO,  # Додаємо прокрутку, якщо контент
більший за екран
            spacing=15)
    )

    refresh_user_list()  # Ініціалізуємо початкове завантаження користувачів
```

Файл auth_page.py.py

```
from db import get_user_by_username, create_user, verify_password

# Відповідність назви ролі до її ID в базі даних
ROLE_MAP = {
    "user": 1,
    "admin": 2
}


def get_auth_page(on_login_success):
    # --- Стан ---
```

```
    mode = {"value": "login"}  # або "register"

    # --- Елементи інтерфейсу ---
    username_field = ft.TextField(label="Ім’я користувача", color="#FFFFFF",
border_color="#FFD700",
                                  label_style=ft.TextStyle(color="#B0B0B0"))
    password_field = ft.TextField(label="Пароль", password=True,
can_reveal_password=True, color="#FFFFFF",
                                  border_color="#FFD700",
label_style=ft.TextStyle(color="#B0B0B0"))
    message_text = ft.Text("", color="red", size=20)

    role_radio = ft.RadioGroup(
        content=ft.Row([
            ft.Text("Оберіть роль:", color="white",
weight=ft.FontWeight.BOLD, size=25),
            ft.Radio(value="user", label="Користувач", fill_color="white",
label_style=ft.TextStyle(color="#B0B0B0")),
            ft.Radio(value="admin", label="Адміністратор",
fill_color="white",
                     label_style=ft.TextStyle(color="#B0B0B0")),
        ]),
        value="user",
        visible=False  # спочатку приховано (для реєстрації)
    )

    # --- Кнопки ---
    login_button = ft.ElevatedButton("Увійти", on_click=lambda e:
handle_login(e), visible=True)
    register_button = ft.ElevatedButton("Зареєструватися", on_click=lambda e:
handle_register(e), visible=False)

    def switch_mode(e, new_mode):
        mode["value"] = new_mode
        is_register = new_mode == "register"
        role_radio.visible = is_register
        login_button.visible = not is_register
        register_button.visible = is_register
        message_text.value = ""
        e.page.update()

    # --- Логіка входу ---
    def handle_login(e):
        username = username_field.value.strip()
        password = password_field.value.strip()

        if not username or not password:
            message_text.value = "Заповніть усі поля"
            e.page.update()
            return

        user = get_user_by_username(username)
        if user and verify_password(password, user["password_hash"]):
            role_id = user.get("role_id", 1)
```

```
            role_name = [name for name, id in ROLE_MAP.items() if id ==
role_id][0]

            e.page.session.set("user_id", user["id"])
            e.page.session.set("username", username)
            e.page.session.set("role", role_name)

            if role_name == "admin":
                e.page.session.set("next_route", "/admin_dashboard")
            else:
                e.page.session.set("next_route", "/user_dashboard")

            on_login_success(user["id"])
        else:
            message_text.value = "Невірне ім’я користувача або пароль"
            e.page.update()

    # --- Логіка реєстрації ---
    def handle_register(e):
        username = username_field.value.strip()
        password = password_field.value.strip()
        selected_role = role_radio.value

        if not username or not password:
            message_text.value = "Заповніть усі поля"
            e.page.update()
            return

        if get_user_by_username(username):
            message_text.value = "Користувач уже існує"
        else:
            role_id = ROLE_MAP.get(selected_role, 1)
            create_user(username, password, role_id)
            message_text.value = "Успішна реєстрація! Тепер увійдіть."
            # автоматичне повернення до логіну
            switch_mode(e, "login")

        e.page.update()

    # --- Головна форма ---
    return ft.Column(
        [
            ft.Text("Вхід / Реєстрація", size=24, weight=ft.FontWeight.BOLD,
color="#FFD700"),

            # Кнопки перемикання режимів
            ft.Row(
                [
                    ft.ElevatedButton(
                        "Увійти",
                        on_click=lambda e: switch_mode(e, "login"),
                        bgcolor="#FFD700",
                        color="black",
```

```
                        style=ft.ButtonStyle(
                            shape=ft.RoundedRectangleBorder(radius=20),
                            elevation=5,
                        )
                    ),
                    ft.Text("/"),
                    ft.ElevatedButton(
                        "Зареєструватися",
                        on_click=lambda e: switch_mode(e, "register"),
                        bgcolor="#FFD700",
                        color="black",
                        style=ft.ButtonStyle(
                            shape=ft.RoundedRectangleBorder(radius=20),
                            elevation=5,
                        )
                    ),
                ],
                spacing=10,
                alignment=ft.MainAxisAlignment.CENTER  # Центрування кнопок
по горизонталі
            ),

            username_field,
            password_field,
            role_radio,

            # Кнопки підтвердження
            ft.Row([
                login_button,
                register_button
            ], spacing=10),

            message_text
        ],
        spacing=20,
        alignment=ft.MainAxisAlignment.CENTER,
        horizontal_alignment=ft.CrossAxisAlignment.CENTER,
    )
```