Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)

«_____»_____202_ р.

КВАЛІФІКАЦІЙНА РОБОТА
на тему

«РОЗРОБКА ТЕЛЕГРАМ БОТА ДЛЯ ВДОСКОНАЛЕННЯ НАВИКІВ ПРОГРАМУВАННЯ»

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавр

Виконавець роботи Хряпченко Павло Романович
_____«____» _____202_ р.
(підпис)

Науковий керівник доцент, к.ф.-м.н. Черненко О. О.
_____«____»_____202_ р.
(підпис)

Рецензент

ПОЛТАВА 2026

РЕФЕРАТ

Записка: 71 с., 12 рис., 2 таблиці, 1 додаток, 14 джерел.

ТЕЛЕГРАМ-БОТ, МІКРОНАВЧАННЯ, ПРОГРАМУВАННЯ, ГЕЙМІФІКАЦІЯ, PYTHON, AIOGRAM, POSTGRESQL, ІНТЕРВАЛЬНЕ ПОВТОРЕННЯ, FSM

Об'єктом розробки є програмне забезпечення — Telegram-бот для вдосконалення навичок програмування у форматі мікронавчання.

Предметом розробки є програмна реалізація серверної логіки бота, навчального контенту та механізмів гейміфікації на основі мови програмування Python.

Метою роботи є створення Telegram-бота, який знижує поріг входу до практики програмування завдяки коротким інтерактивним урокам і формує звичку щоденних занять через гейміфікацію.

Результатом роботи стало розроблення програмного засобу «CodeQuest» на базі мови Python з використанням фреймворку aiogram, реляційної СУБД PostgreSQL та сховища Redis. Реалізовано ключові модулі:

- модуль реєстрації та онбордингу — створення облікового запису, вибір мови програмування, рівня та щоденної цілі;
- модуль курсів та уроків — каталог курсів, послідовне відкриття уроків, облік прогресу;
- рушій уроків — проходження уроку як скінченного автомата з підтримкою восьми типів інтерактивних вправ;
- модуль гейміфікації — нарахування досвіду, рівні, звання, досягнення та серія днів;
- модуль щоденної цілі та челенджу дня — облік цілі, окремий урок дня з бонусом;
- модуль рейтингу — глобальний та тижневий рейтинги гравців;
- модуль інтервального повторення — повернення помилкових вправ для закріплення;
- модуль планувальника — нагадування, генерація челенджів та скидання

тижневого рейтингу.

Особливості: повністю асинхронна архітектура, взаємодія виключно через екранні кнопки без набору коду, навчальний контент для п'яти мов програмування, контейнеризація на базі Docker.

Проведено офлайн-перевірку коректності проєкту: перевірку синтаксису всіх модулів, перевірку імпорту застосунку та реєстрації обробників, перевірку логіки восьми типів вправ і структурної валідності навчального контенту.

«CodeQuest» може бути використаний як засіб самостійного вивчення програмування, як доповнення до навчальних курсів закладів освіти та як інструмент підтримання набутих навичок.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
API	Application Programming Interface — програмний інтерфейс застосунку
CRUD	Create, Read, Update, Delete — базові операції над даними
DDL	Data Definition Language — мова визначення даних
FSM	Finite State Machine — скінченний автомат (машина станів)
HTML	HyperText Markup Language — мова розмітки гіпертексту
HTTP	HyperText Transfer Protocol — протокол передачі гіпертексту
JSON	JavaScript Object Notation — текстовий формат обміну даними
JSONB	Binary JSON — двійковий формат зберігання JSON у PostgreSQL
ORM	Object-Relational Mapping — об'єктно-реляційне відображення
REST	Representational State Transfer — архітектурний стиль взаємодії
SQL	Structured Query Language — мова структурованих запитів
TTL	Time To Live — час життя запису
UI	User Interface — інтерфейс користувача
UX	User Experience — досвід взаємодії користувача
XP	Experience Points — бали досвіду користувача

БД	База даних
ООП	Об'єктно-орієнтоване програмування
СУБД	Система управління базами даних

ВСТУП

Програмування є однією з ключових професійних компетенцій сучасного цифрового суспільства, а попит на кваліфікованих фахівців у галузі інформаційних технологій продовжує зростати швидше, ніж пропозиція. Опанування навичок програмування — процес тривалий, який вимагає не стільки одноразового засвоєння теоретичного матеріалу, скільки систематичної та регулярної практики. Саме регулярність занять, а не їх інтенсивність, є визначальним чинником успішного формування стійких практичних навичок.

Водночас на практиці саме регулярність є найслабшою ланкою самостійного навчання. Без зовнішньої мотивації та структурованого плану більшість людей, які розпочинають вивчати програмування самостійно, займаються нерегулярно і з часом припиняють навчання. Класичні платформи для практики програмування, такі як LeetCode чи HackerRank, орієнтовані на тривалі сесії за комп'ютером із написанням повноцінного коду, що погано вписується у щільний розклад студента чи працівника і не дозволяє використовувати для навчання короткі проміжки вільного часу протягом дня.

Альтернативою класичному підходу є концепція мікронавчання (microlearning) — подання навчального матеріалу невеликими, логічно завершеними порціями, проходження яких займає кілька хвилин. Мікронавчання добре поєднується з принципами гейміфікації — застосування ігрових елементів (балів, рівнів, досягнень, рейтингів) у неігровому контексті — та з ідеєю формування звички через короткі щоденні дії. Цей підхід довів свою ефективність у застосунках для вивчення іноземних мов, найвідомішим з яких є Duolingo.

Зручною платформою для реалізації навчального інструменту у форматі мікронавчання є месенджер Telegram та його програмний інтерфейс для ботів. Telegram-бот не потребує встановлення окремого застосунку, доступний на всіх платформах, завжди під рукою у користувача і має вбудований канал сповіщень, що є ефективним засобом нагадувань для формування звички. Мова програмування

Python разом із сучасним асинхронним фреймворком aiogram дають змогу побудувати продуктивний та масштабований бот із порівняно компактною кодовою базою.

Метою кваліфікаційної роботи є розробка Telegram-бота для вдосконалення навичок програмування, який знижує поріг входу до практики завдяки коротким інтерактивним урокам, забезпечує миттєвий зворотний зв'язок щодо кожної вправи та формує звичку щоденних занять за допомогою гейміфікації й нагадувань.

Для досягнення поставленої мети у роботі визначено такі основні завдання:

- проаналізувати предметну область мікронавчання програмування, дослідити принципи гейміфікації, інтервального повторення та формування навчальної звички;
- провести огляд існуючих рішень для навчання програмування та обґрунтувати доцільність розробки власного Telegram-бота;
- дослідити теоретичні засади побудови Telegram-ботів: програмний інтерфейс Telegram Bot API, фреймворк aiogram, асинхронну модель виконання, реляційну модель даних та механізм скінченних автоматів;
- обґрунтувати вибір технологічного стеку та спроєктувати багатошарову архітектуру системи;
- спроєктувати реляційну схему бази даних для зберігання користувачів, навчального контенту та прогресу;
- реалізувати ключові модулі бота: онбординг, рушій уроків з вісьмома типами вправ, гейміфікацію, рейтинг, щоденну ціль, челендж дня та інтервальне повторення;
- підготувати навчальний контент і конфігурацію для контейнерного розгортання, провести перевірку коректності реалізованих компонентів.

Об'єктом дослідження є процес самостійного вивчення та вдосконалення навичок програмування у форматі мікронавчання.

Предметом дослідження є програмна реалізація Telegram-бота на основі мови Python, що включає рушій інтерактивних уроків, систему гейміфікації, інтервальне повторення та керування навчальним контентом.

Методи дослідження. У роботі застосовано методи системного аналізу для дослідження предметної області, методи об'єктно-орієнтованого проєктування для побудови архітектури, методи реляційної алгебри та нормалізації для проєктування схеми даних, а також практичні методи розробки програмного забезпечення з використанням контейнеризації.

Практичне значення одержаних результатів полягає у створенні готового до використання навчального інструменту, який може застосовуватися для самостійного вивчення програмування, як доповнення до освітніх курсів та для підтримання набутих навичок.

ПОСТАНОВКА ЗАДАЧІ

Основною задачею кваліфікаційної роботи є проєктування та реалізація Telegram-бота для вдосконалення навичок програмування, який забезпечує навчання у форматі коротких інтерактивних уроків і підтримує регулярність занять засобами гейміфікації.

Розроблюваний бот повинен надавати користувачу структурований навчальний контент у вигляді курсів та уроків, забезпечувати проходження уроків як послідовності інтерактивних вправ, виконуваних виключно за допомогою екранних кнопок без набору коду з клавіатури, а також зберігати персональний прогрес кожного користувача у реляційній СУБД.

Функціональні вимоги до системи передбачають наявність модулів реєстрації та онбордингу, каталогу курсів і уроків, рушія уроків з підтримкою восьми типів вправ, гейміфікації (досвід, рівні, звання, досягнення, серія), щоденної цілі, челенджу дня, рейтингу, інтервального повторення, нагадувань та налаштувань. Нефункціональні вимоги охоплюють асинхронну обробку запитів для забезпечення продуктивності за наявності багатьох одночасних користувачів, стійкість до помилок, простоту розгортання засобами контейнеризації та можливість розширення навчального контенту без зміни програмного коду.

Для досягнення поставленої задачі необхідно виконати такі підзадачі:

1. проаналізувати предметну область мікронавчання програмування, дослідити принципи гейміфікації та формування навчальної звички, виявити вимоги до сучасних навчальних інструментів;
2. провести огляд існуючих рішень для навчання програмування, виконати порівняльний аналіз їх можливостей та скласти таблицю порівняння аналогів;
3. дослідити теоретичні засади побудови Telegram-ботів: програмний інтерфейс Telegram Bot API, можливості фреймворку aiogram, асинхронну модель виконання мови Python та механізм скінченних автоматів для керування діалогом;

4. обґрунтувати вибір технологічного стеку: версії мови Python, бот-фреймворку, реляційної СУБД, сховища станів, бібліотеки доступу до даних та засобів розгортання;
5. спроєктувати багатошарову архітектуру системи з чітким розподілом обов'язків між шарами проміжного програмного забезпечення, обробників, сервісів та репозиторіїв;
6. спроєктувати реляційну схему бази даних для зберігання користувачів, курсів, уроків, вправ, прогресу проходження, досягнень та черги повторення;
7. реалізувати модуль реєстрації нового користувача та онбордингу з вибором мови програмування, рівня підготовки та щоденної цілі;
8. реалізувати рушій уроків на основі скінченного автомата зі станом у сховищі Redis та підтримкою восьми типів інтерактивних вправ;
9. реалізувати модуль гейміфікації: нарахування досвіду, перерахунок рівнів і звань, систему досягнень та облік серії днів;
10. реалізувати модулі щоденної цілі, челенджу дня та рейтингу (глобального і тижневого);
11. реалізувати модуль інтервального повторення, що повертає помилкові вправи на повторне опрацювання за зростаючими інтервалами;
12. реалізувати фоновий планувальник для надсилання нагадувань, генерації челенджів дня та періодичного скидання тижневого рейтингу;
13. підготувати навчальний контент для кількох мов програмування та конфігурацію для контейнерного розгортання системи;
14. провести перевірку коректності реалізованих компонентів та оформити інструкцію для користувача.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1. Аналіз предметної області мікронавчання програмування

Навчання програмуванню принципово відрізняється від засвоєння суто теоретичних дисциплін. Програмування є практичною навичкою, і її формування підпорядковується тим самим закономірностям, що й опанування будь-якого складного вміння: воно вимагає багаторазового повторення, систематичної практики та своєчасного зворотного зв'язку. Дослідження у галузі педагогіки та когнітивної психології послідовно підтверджують, що розподілена у часі практика (distributed practice) є значно ефективнішою за концентроване, але епізодичне навчання.

Ключовою проблемою самостійного вивчення програмування є не брак навчальних матеріалів, яких сьогодні існує надлишок, а брак регулярності та мотивації. Людина, яка навчається самостійно, не має зовнішньої структури — розкладу занять, викладача, групи — і змушена покладатися виключно на власну дисципліну. За відсутності негайної винагороди й помітного прогресу мотивація швидко згасає, а заняття стають усе рідшими, доки не припиняються зовсім.

Концепція мікронавчання пропонує розв'язання цієї проблеми через подання матеріалу невеликими, логічно завершеними порціями. Мікроурок триває кілька хвилин і не вимагає від користувача виділення значного часу та концентрації, тому його легко вмонтувати у звичайний день — у дорозі, у черзі, під час перерви. Низький поріг входу — головна перевага мікронавчання: щоб розпочати заняття, не потрібно «налаштовуватися», достатньо відкрити застосунок. Узагальнену схему процесу мікронавчання у месенджері наведено на рисунку (див. рис. 1.1).

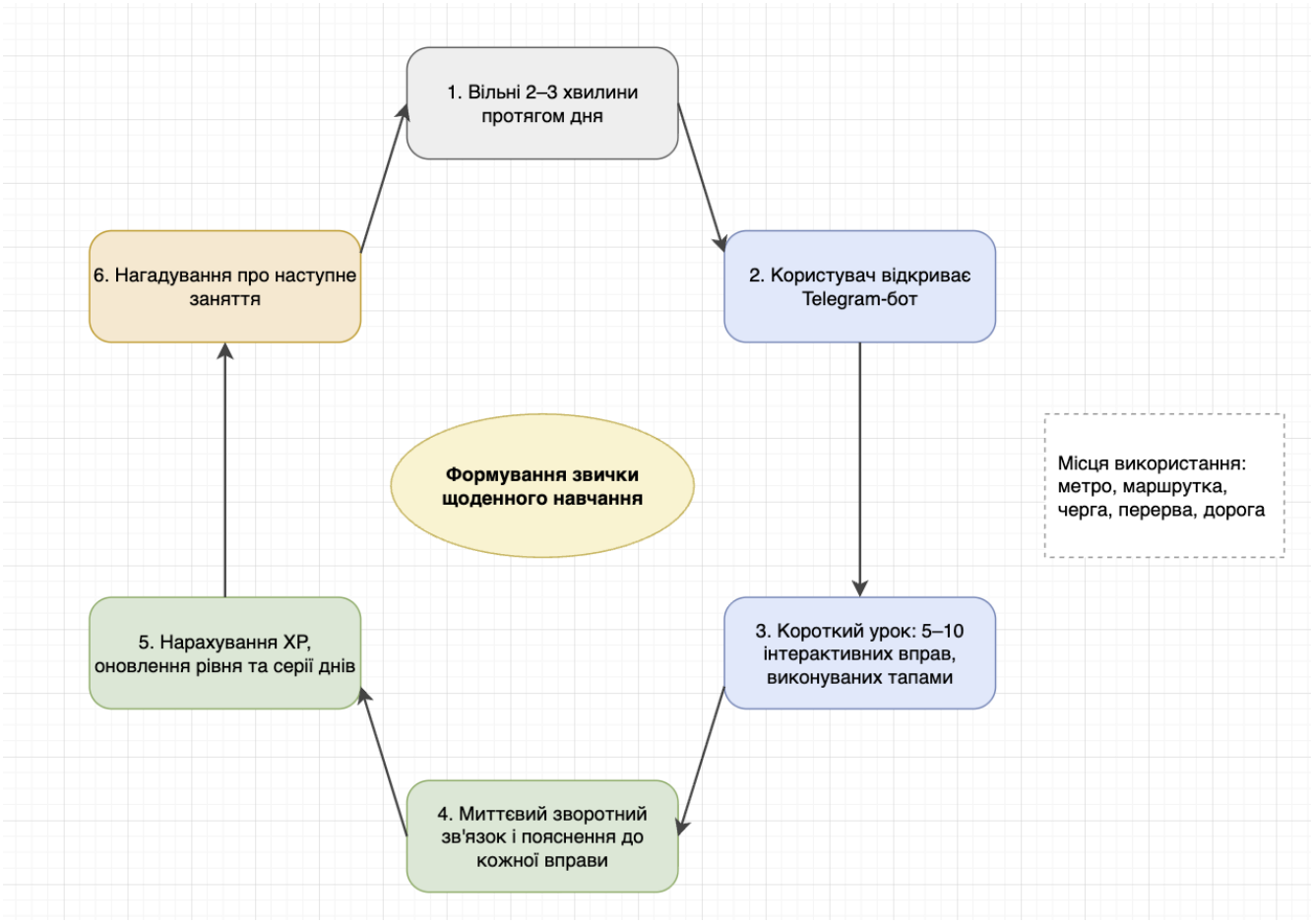

Рисунок 1.1 – Принцип мікронавчання: коротка навчальна сесія в месенджері

Мікронавчання набуває найбільшої ефективності у поєднанні з гейміфікацією. Гейміфікація — це застосування характерних для ігор елементів та механік у неігровому контексті з метою підвищення залученості користувача. До типових ігрових елементів належать бали досвіду, рівні, звання, досягнення (бейджі), смуги прогресу, рейтинги та серії. Психологічна основа дієвості гейміфікації полягає у задоволенні базових потреб людини у відчутті компетентності, автономії та поступу: видимий прогрес і регулярна винагорода створюють позитивне підкріплення, що формує звичку повертатися до застосунку.

Окремим важливим механізмом є серія (streak) — лічильник днів поспіль, протягом яких користувач виконував щоденну ціль. Серія використовує психологічний ефект небажання втрачати вже накопичений результат і є одним із найдієвіших інструментів формування звички у навчальних застосунках. У

поєднанні зі своєчасними нагадуваннями серія перетворює абстрактний намір «вчитися регулярно» на конкретну щоденну дію.

Ще одним науково обґрунтованим методом, релевантним для предметної області, є інтервальне повторення (spaced repetition). Цей метод ґрунтується на ефекті інтервалу: матеріал, повторюваний через зростаючі проміжки часу, запам'ятовується значно міцніше, ніж під час суцільного повторення. У навчальному застосунку інтервальне повторення доцільно застосовувати саме до тих вправ, у яких користувач помилявся, повертаючи їх на повторне опрацювання у певні моменти часу для надійного закріплення слабких місць.

Таким чином, аналіз предметної області показує, що ефективний інструмент для вдосконалення навичок програмування має поєднувати три складові: мікронавчання як спосіб зниження порога входу, гейміфікацію як механізм підтримання мотивації та регулярності, а також інтервальне повторення як засіб надійного закріплення матеріалу. Саме на цих засадах побудовано розроблюваний у роботі Telegram-бот.

1.2. Огляд існуючих рішень для навчання програмування

На ринку освітніх продуктів представлено широкий спектр рішень для навчання програмування, які можна умовно поділити на три категорії: мобільні застосунки мікронавчання, веб-платформи для практики алгоритмічних задач та навчальні боти у месенджерах. Розглянемо найбільш показові приклади кожної категорії, щоб виявити їх сильні та слабкі сторони.

SoloLearn — один із найпопулярніших мобільних застосунків для навчання програмування. Він пропонує курси з багатьох мов, інтерактивні уроки, систему досягнень, рейтинги та активну спільноту. Сильними сторонами SoloLearn є велика база контенту та розвинена соціальна складова. Водночас застосунок потребує встановлення, значна частина контенту доступна лише за платною підпискою, а інтерфейс не локалізовано українською мовою.

Mimo — мобільний застосунок, що позиціонує себе саме як інструмент мікронавчання програмування. Mimo подає матеріал короткими уроками, активно використовує гейміфікацію та формування звички через щоденні цілі й серії. Недоліками є переважно платна модель доступу, орієнтація на мобільну платформу з необхідністю встановлення та відсутність української локалізації.

LeetCode — провідна веб-платформа для практики алгоритмічних задач та підготовки до технічних співбесід. Вона містить велику базу задач різної складності та середовище виконання коду. LeetCode орієнтована на досвідчених користувачів і тривалі сесії за комп'ютером із написанням повноцінного коду, тому не підходить для мікронавчання та має високий поріг входу для початківців.

Навчальні Telegram-боти — окрема категорія рішень, що використовують месенджер як платформу доставлення контенту. Більшість наявних навчальних ботів обмежуються надсиланням теоретичних матеріалів або простих тестів і не реалізують повноцінного рушія інтерактивних уроків, гейміфікації та інтервального повторення. Це залишає нішу для якісного навчального бота, який поєднує переваги месенджера з повноцінною навчальною механікою.

Для систематизованого порівняння розглянутих рішень за ключовими критеріями складено таблицю 1.1. Розроблюваний у межах цієї кваліфікаційної роботи продукт «CodeQuest» включено до таблиці для подальшого обґрунтування його місця серед існуючих рішень.

Таблиця 1.1 — Порівняння рішень для навчання програмування

Критерій	**SoloLearn**	**Mimo**	**LeetCode**	**CodeQuest**
Формат доставлення	застосунок	застосунок	веб-платформа	Telegram-бот
Мікронавчання	+	+	–	+
Навчання без набору коду	частково	частково	–	+
Гейміфікація	+	+	частково	+
Інтервальне повторення	–	частково	–	+
Серія та щоденна ціль	+	+	–	+
Челендж дня	–	+	щоденна задача	+
Потрібне встановлення	+	+	–	–
Безкоштовний повний доступ	–	–	частково	+
Українська локалізація	–	–	–	+

Аналіз таблиці 1.1 показує, що наявні рішення мають вагомі переваги — велику базу контенту, розвинену спільноту, зрілість продукту, — проте жодне з них не поєднує одночасно формат мікронавчання у месенджері, відсутність потреби встановлення, повноцінну навчальну механіку з інтервальним повторенням та українську локалізацію. Саме це поєднання визначає нішу для розроблюваного продукту.

1.3. Обґрунтування доцільності власної розробки

Проведений у попередніх підрозділах аналіз дозволяє зробити висновок, що жодне з існуючих рішень не є оптимальним для сценарію, на який орієнтована ця робота, — навчання програмуванню короткими щоденними сесіями безпосередньо у месенджері, без потреби у встановленні окремого застосунку та без фінансового бар'єру. Мобільні застосунки мікронавчання вимагають встановлення і здебільшого є платними; веб-платформи орієнтовані на тривалі сесії та досвідчених користувачів; наявні навчальні боти не реалізують повноцінної навчальної механіки.

Вибір саме Telegram-бота як форми реалізації навчального інструменту обґрунтований кількома чинниками. По-перше, месенджер уже встановлено на пристроях переважної більшості потенційних користувачів, тому поріг входу мінімальний — не потрібно завантажувати й встановлювати окремий застосунок. По-друге, месенджер завжди під рукою, що ідеально відповідає ідеї коротких сесій у дорозі. По-третє, Telegram надає вбудований канал сповіщень, який є ефективним і ненав'язливим засобом нагадувань. По-четверте, бот працює однаково на всіх платформах — Android, iOS, настільних операційних системах і у браузері — без окремої розробки під кожну з них.

Свідомим проєктним рішенням є відмова від виконання довільного коду користувача у пісочниці. У розроблюваному боті користувач не пише код, а оперує готовими фрагментами: обирає відповідь, упорядковує рядки, вставляє пропущений

токен, знаходить помилковий рядок. Кожна вправа має наперед визначену скінченну множину правильних відповідей, тому правильність перевіряється миттєвим зіставленням відповіді користувача з еталоном, без потреби у зовнішніх сервісах виконання коду. Це суттєво спрощує архітектуру, прибирає привілейовані контейнери та робить розгортання надійним на будь-якій платформі. Водночас навчальна цінність зберігається повністю: користувач читає, аналізує та добудовує реальний код.

Власна розробка дає повний контроль над навчальною механікою та контентом, дозволяє безкоштовно надати користувачам повний доступ, локалізувати інтерфейс українською мовою та реалізувати саме той набір механік — мікронавчання, гейміфікацію та інтервальне повторення, — який є оптимальним для поставленої мети. З огляду на викладене, розробка власного Telegram-бота для вдосконалення навичок програмування є доцільною та обґрунтованою.

2. ТЕОРЕТИЧНА ЧАСТИНА

2.1. Telegram Bot API та фреймворк aiogram

Telegram-бот — це спеціальний обліковий запис у месенджері Telegram, керований не людиною, а програмою. Уся взаємодія програми з месенджером здійснюється через Telegram Bot API — програмний інтерфейс на основі протоколу HTTP, що надається компанією Telegram. Бот реєструється через службового бота @BotFather, який видає унікальний токен — рядок, що використовується для автентифікації всіх запитів до Bot API [3].

Telegram Bot API передбачає два способи отримання оновлень (updates) — повідомлень про нові події, адресовані боту. Перший спосіб — long polling — полягає у тому, що застосунок самостійно періодично звертається до сервера Telegram із запитом на нові оновлення, причому сервер утримує з'єднання відкритим до появи нових подій. Другий спосіб — webhook — передбачає, що сервер Telegram сам надсилає оновлення на наперед заданий HTTP-адрес застосунку. Long polling простіший у налаштуванні, не потребує публічного доменного імені та сертифіката, тому є зручним для розробки і для невеликих та середніх ботів; саме його застосовано у розроблюваній системі.

Оновлення в Telegram Bot API представлені кількома типами, найважливішими з яких для інтерактивного бота є повідомлення (message) та натискання кнопки вбудованої клавіатури (callback query). Вбудована клавіатура (inline keyboard) — це набір кнопок, прикріплених безпосередньо до повідомлення бота; кожна кнопка несе рядок корисних даних (callback data), який повертається боту при натисканні. Саме вбудовані клавіатури є основним засобом взаємодії у розроблюваному боті, оскільки дозволяють користувачу виконувати вправи натисканнями, без набору тексту.

Безпосередня робота з Bot API через формування HTTP-запитів є трудомісткою, тому на практиці використовують бот-фреймворки. Для мови Python

таким фреймворком є aiogram — сучасна асинхронна бібліотека для розробки Telegram-ботів. У роботі використано aiogram третьої версії, яка надає чітку та послідовну архітектуру побудови ботів. Ключовими поняттями aiogram є: об'єкт Bot, що інкапсулює виклики Bot API; диспетчер Dispatcher, що приймає оновлення та маршрутизує їх; роутери Router, які групують пов'язані обробники; обробники (handlers) — асинхронні функції, прив'язані до певного типу подій за допомогою фільтрів [2].

Окремою сильною стороною aiogram є система фільтрів та фабрик callback-даних. Фабрика callback-даних дає змогу описати структуру корисних даних кнопки у вигляді класу з полями, автоматично серіалізувати її у компактний рядок і десеріалізувати назад при натисканні, із суворою типізацією. Це робить код обробників декларативним і захищеним від помилок розбору рядків. Узагальнений життєвий цикл оновлення у застосунку на основі aiogram наведено на рисунку (див. рис. 2.1).

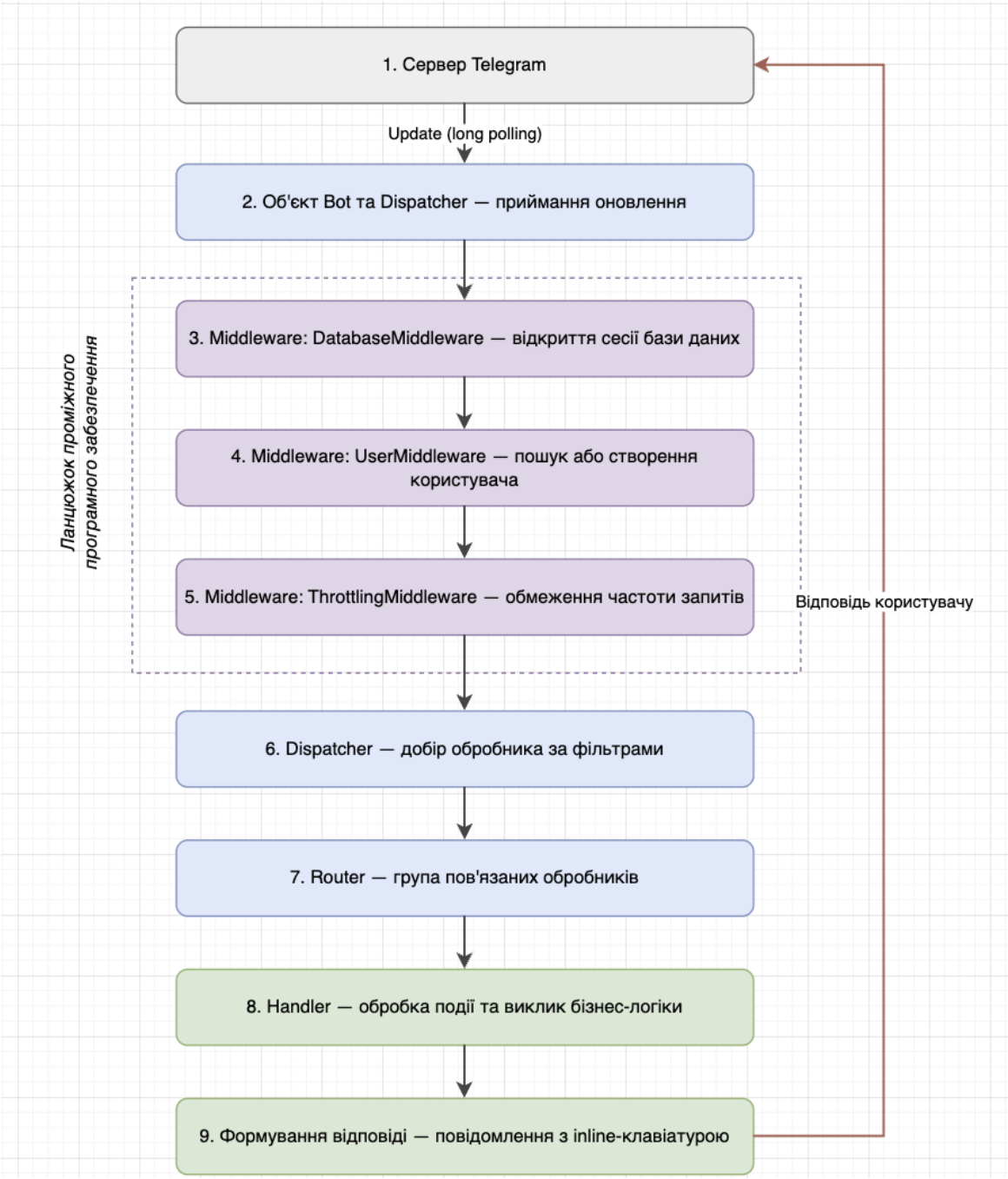

Рисунок 2.1 – Життєвий цикл оновлення у фреймворку aiogram

Як видно зі схеми, оновлення послідовно проходить ланцюжок проміжного програмного забезпечення (middleware), після чого диспетчер визначає відповідний обробник за фільтрами і викликає його. Така модель забезпечує чисте відокремлення наскрізної логіки від основної логіки обробників та є основою для побудови шарової архітектури застосунку, що розглядається у наступному підрозділі.

2.2. Асинхронна модель виконання та шарова організація коду

Telegram-бот за своєю природою є застосунком, що переважну частину часу очікує — на оновлення від сервера Telegram, на відповідь бази даних, на завершення мережевих операцій. Якщо обробляти запити синхронно й послідовно, бот зможе обслуговувати лише одного користувача в один момент часу, а решта чекатимуть. Розв'язанням цієї проблеми є асинхронна модель виконання, реалізована у мові Python через бібліотеку asyncio та синтаксичні конструкції async і await [1, 11].

Сутність асинхронної моделі полягає у тому, що замість блокування потоку виконання на час очікування операції введення-виведення керування передається циклу подій (event loop), який у цей час виконує інші готові до роботи задачі. Коли очікувана операція завершується, виконання відповідної задачі поновлюється. Таким чином, один потік виконання здатний обслуговувати велику кількість користувачів одночасно, ефективно використовуючи проміжки очікування. Асинхронна функція позначається ключовим словом async, а виклик іншої асинхронної операції — словом await; саме у точках await цикл подій може перемкнутися на іншу задачу.

Фреймворк aiogram повністю побудований на асинхронній моделі: усі обробники є асинхронними функціями. Це накладає вимогу асинхронності й на решту компонентів системи — доступ до бази даних, до сховища станів, до зовнішніх сервісів. Тому в роботі застосовано асинхронний драйвер бази даних та асинхронний клієнт сховища станів, що забезпечує наскрізну неблокуючу обробку запитів.

Для того щоб кодова база залишалася зрозумілою та придатною до супроводу при зростанні функціональності, застосунок організовано за принципом шарової архітектури. Кожен шар відповідає за чітко визначене коло задач і взаємодіє лише із сусіднім шаром, що знаходиться нижче. Узагальнену шарову архітектуру розроблюваного бота наведено на рисунку (див. рис. 2.2).

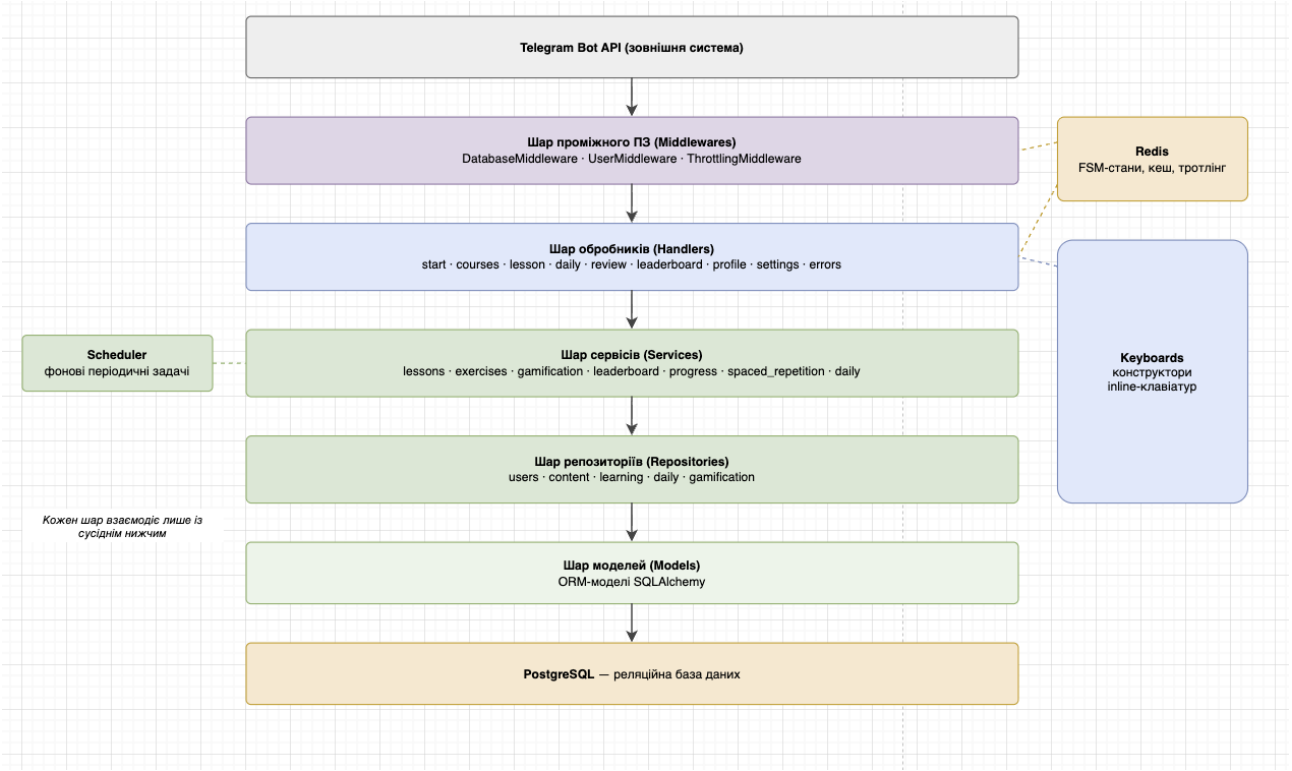

Рисунок 2.2 – Шарова архітектура застосунку CodeQuest

Виокремлюють такі шари: проміжне програмне забезпечення (middleware) реалізує наскрізну логіку — створення сесії бази даних, реєстрацію користувача, обмеження частоти запитів; обробники (handlers) приймають оновлення та формують відповіді; сервіси (services) містять бізнес-логіку — рушій уроків, перевірку вправ, нарахування досвіду, гейміфікацію; репозиторії (repositories) ізолюють увесь доступ до бази даних; моделі (models) описують структуру даних. Перевагою такого поділу є тестованість окремих шарів, локалізація змін та зрозумілість коду: щоб знайти певну логіку, розробник одразу знає, у якому шарі її шукати.

2.3. Реляційна модель даних та об'єктно-реляційне відображення

Telegram-бот для навчання має зберігати значний обсяг структурованих даних — облікові записи користувачів, навчальний контент, прогрес проходження уроків, досягнення, чергу повторення. Ці дані характеризуються чіткою структурою та численними зв'язками між собою, тому оптимальним сховищем для них є реляційна СУБД. У роботі використано PostgreSQL — одну з найпотужніших реляційних СУБД з відкритим кодом, що повністю відповідає стандарту SQL, підтримує транзакції з повним набором властивостей ACID та має розширений набір типів даних [5].

Реляційна модель подає дані у вигляді сукупності таблиць (відношень), кожна з яких складається з рядків (записів) та стовпців (атрибутів). Зв'язки між таблицями встановлюються через зовнішні ключі — атрибути, що посилаються на первинний ключ іншої таблиці. Для уникнення надмірності та аномалій оновлення схему даних приводять до нормальних форм; на практиці зазвичай достатньо третьої нормальної форми, яка усуває транзитивні залежності атрибутів від первинного ключа.

Окремою важливою можливістю PostgreSQL, використаною у роботі, є тип даних JSONB — двійковий формат зберігання документів JSON. Тип JSONB дозволяє зберігати у стовпці таблиці структуру змінного складу, що ідеально підходить для навчальних вправ: вісім різних типів вправ мають різну внутрішню структуру даних, але всі зберігаються в одній таблиці завдяки JSONB-стовпцям для вмісту вправи та еталонної відповіді. Це поєднує переваги реляційної моделі для основної структури з гнучкістю документної моделі для змінної частини.

Безпосередня робота з базою даних мовою SQL із застосунку призводить до змішування коду доступу до даних з бізнес-логікою та до дублювання. Розв'язанням є технологія об'єктно-реляційного відображення (ORM), яка встановлює відповідність між таблицями бази даних і класами мови програмування, а між рядками таблиць — і об'єктами цих класів. У роботі застосовано бібліотеку SQLAlchemy у версії 2.0, що є де-факто стандартом ORM для мови Python.

SQLAlchemy дозволяє описати кожну таблицю як клас-модель, успадкований від спільного базового класу, де стовпці оголошуються як типізовані атрибути. На основі цього опису бібліотека здатна автоматично згенерувати команди визначення даних та створити схему бази даних. Запити до бази формуються виразами мовою Python, які SQLAlchemy транслює у SQL. У роботі використано асинхронний режим SQLAlchemy у поєднанні з асинхронним драйвером asyncpg, що забезпечує неблокуючий доступ до PostgreSQL та узгоджується із загальною асинхронною моделлю застосунку. Створення схеми бази даних безпосередньо з моделей при старті застосунку звільняє від потреби в окремих сценаріях міграцій для проєкту такого масштабу [4, 6].

2.4. Скінченні автомати та алгоритм інтервального повторення

Проходження уроку в навчальному боті є багатокроковим діалогом: користувач послідовно виконує вправи, і на кожному кроці бот повинен пам'ятати, який саме урок проходиться, яка вправа є поточною, скільки відповідей уже надано правильно. Така взаємодія з пам'яттю про попередні кроки природно описується математичною моделлю скінченного автомата (Finite State Machine, FSM).

Скінченний автомат — це абстрактна обчислювальна модель, що в кожен момент часу перебуває в одному зі скінченної множини станів і переходить з одного стану в інший у відповідь на вхідні події. У контексті Telegram-бота стан автомата визначає, у якій фазі діалогу перебуває користувач, а до стану прив'язуються довільні дані — так звані дані стану. Фреймворк aiogram надає вбудований механізм FSM: стани оголошуються як класи, а перехід між ними та збереження даних стану виконуються через спеціальний контекст.

Принципово важливим є питання, де зберігати стан користувача. Зберігання у пам'яті процесу є простим, але втрачається при перезапуску бота й унеможливлює горизонтальне масштабування. Тому у роботі стан зберігається у Redis — швидкому сховищі даних типу «ключ — значення», що працює переважно в оперативній пам'яті. Redis забезпечує збереження стану проходження уроку між

окремими натисканнями кнопок, стійкість до перезапусків застосунку та можливість масштабування. Окрім зберігання FSM-станів, Redis у роботі застосовано і для обмеження частоти запитів [7].

Другим теоретичним механізмом, реалізованим у системі, є алгоритм інтервального повторення. Як зазначено у підрозділі 1.1, ефект інтервалу полягає у міцнішому запам'ятовуванні матеріалу, повторюваного через зростаючі проміжки часу. Класичною реалізацією цього принципу є система Лейтнера, у якій картки з матеріалом розподіляються за «коробками» з різними інтервалами повторення: правильна відповідь переводить картку до коробки з більшим інтервалом, помилкова — повертає до початкової.

У розроблюваному боті застосовано спрощену модель інтервального повторення на основі цього принципу. Вправа, у якій користувач помилився, потрапляє до черги повторення з найменшим інтервалом. Кожне успішне повторення збільшує інтервал за наперед заданою послідовністю зростаючих значень (один, три, сім та чотирнадцять днів), а наступна дата повторення обчислюється додаванням поточного інтервалу до сьогоднішньої дати. Після проходження вправи через усі інтервали її вважають закріпленою і вилучають із черги; повторна помилка повертає вправу до початкового інтервалу. Така модель є простою в реалізації, не потребує складних обчислень і водночас реалізує науково обґрунтований принцип розподіленого у часі повторення.

3. ПРАКТИЧНА ЧАСТИНА

3.1. Загальна архітектура системи та структура проєкту

Розроблений програмний засіб «CodeQuest» являє собою Telegram-бот, реалізований мовою Python і запущений як окремий застосунок. Система складається з трьох взаємодійних компонентів: власне застосунку бота, реляційної СУБД PostgreSQL для постійного зберігання даних та сховища Redis для зберігання станів проходження уроків і допоміжних даних. Усі три компоненти розгортаються спільно засобами Docker. Загальну архітектуру системи представлено на рисунку (див. рис. 3.1).

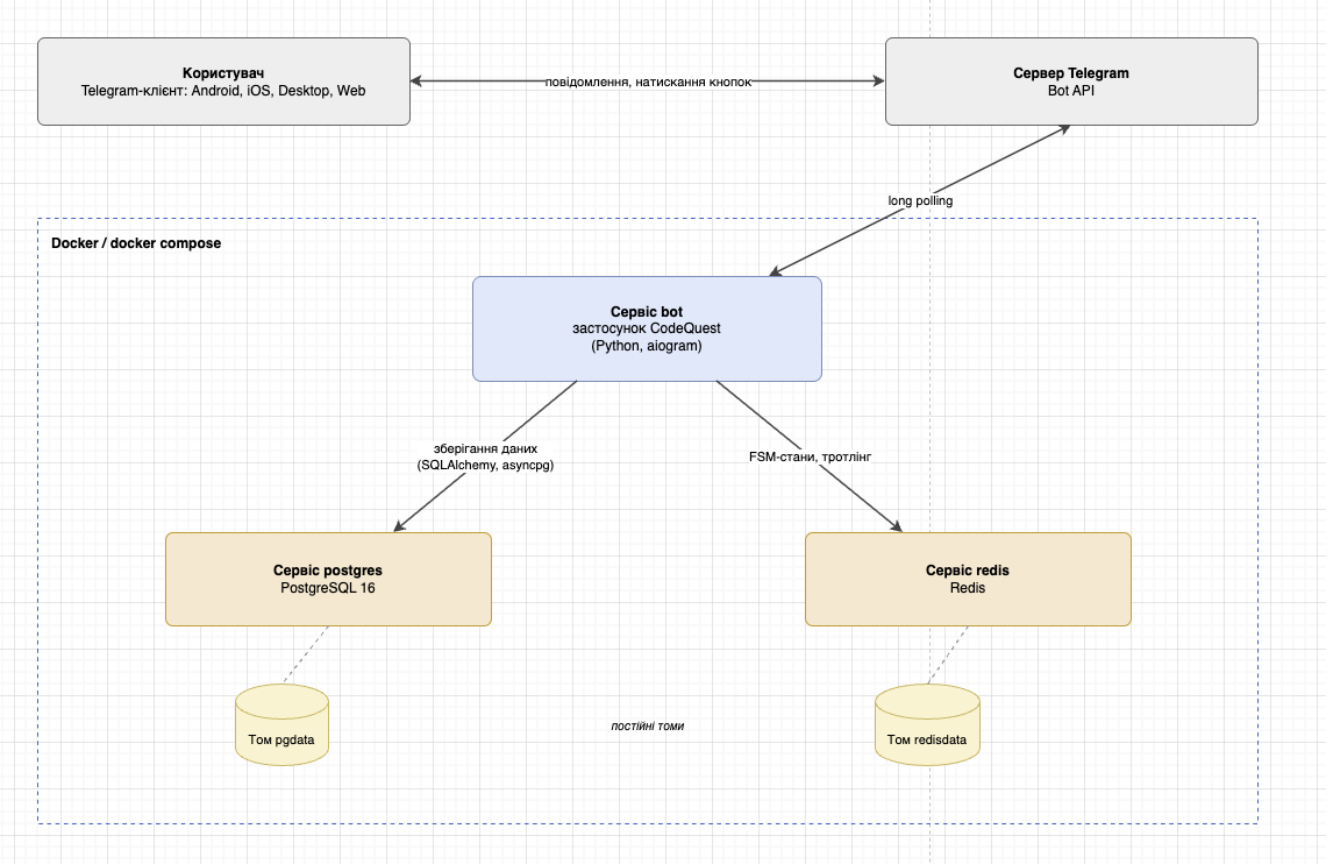

Рисунок 3.1 – Загальна архітектура системи CodeQuest

Застосунок бота взаємодіє з сервером Telegram за схемою long polling, отримуючи оновлення про дії користувачів та надсилаючи у відповідь

повідомлення з вбудованими клавіатурами. Кожне оновлення проходить визначений шлях: ланцюжок проміжного програмного забезпечення, диспетчер, відповідний роутер та обробник, який звертається до сервісного шару, а той — до репозиторіїв і бази даних. Стан проходження уроку зчитується та зберігається у Redis на кожному кроці взаємодії.

Структуру проєкту організовано за принципом шарової архітектури, описаним у підрозділі 2.2, і подано у вигляді набору пакетів, кожен з яких відповідає окремому шару або функціональній підсистемі. Точкою входу є модуль bot.py, який виконує роль композиційного кореня: створює об'єкти бота й диспетчера, реєструє проміжне програмне забезпечення та роутери, ініціалізує планувальник і запускає отримання оновлень. Каталог models містить ORM-моделі даних, database — налаштування підключення до бази, repositories — функції доступу до даних, services — бізнес-логіку, handlers — обробники подій, keyboards — конструктори вбудованих клавіатур, middlewares — проміжне програмне забезпечення, scheduler — фонові задачі, seeds — навчальний контент, utils — допоміжні функції.

Конфігурація застосунку винесена у змінні середовища відповідно до принципу 12-Factor App, що забезпечує незалежність коду від конкретного середовища виконання. Завантаження конфігурації виконує клас на основі бібліотеки pydantic-settings: він зчитує токен бота, параметри підключення до PostgreSQL і Redis із файла середовища або зі змінних оточення та виконує перевірку типів. Конфіденційні дані — насамперед токен бота — не зберігаються у коді й не потрапляють до системи контролю версій [9, 13].

Послідовність запуску застосунку у модулі bot.py має чітко виражений каскад: спочатку створюється схема бази даних з ORM-моделей, далі база наповнюється навчальним контентом, після чого створюється сховище станів на основі Redis, об'єкти бота й диспетчера, реєструються три проміжні шари та дев'ять роутерів, налаштовується й запускається планувальник фонових задач, встановлюється перелік команд бота, і нарешті розпочинається отримання

оновлень. Завершення роботи виконується коректно — з зупинкою планувальника та звільненням з'єднань.

Загальний обсяг кодової бази проєкту становить близько дев'яти тисяч трьохсот рядків коду мовою Python, розподілених між дев'яноста трьома файлами. Така структура забезпечує модульність: кожен файл має чітко окреслену відповідальність, а додавання нового навчального курсу зводиться до створення одного файла з даними і не потребує зміни програмного коду.

3.2. Проєктування схеми бази даних

Реляційну схему бази даних спроєктовано як набір з дванадцяти таблиць, нормалізованих до третьої нормальної форми. Схему створюють безпосередньо з ORM-моделей при старті застосунку засобами SQLAlchemy, без окремих сценаріїв міграцій. Таблиці логічно поділяються на чотири групи: користувачі, навчальний контент, прогрес користувача та гейміфікація. ER-діаграму основних таблиць наведено на рисунку (див. рис. 3.2).

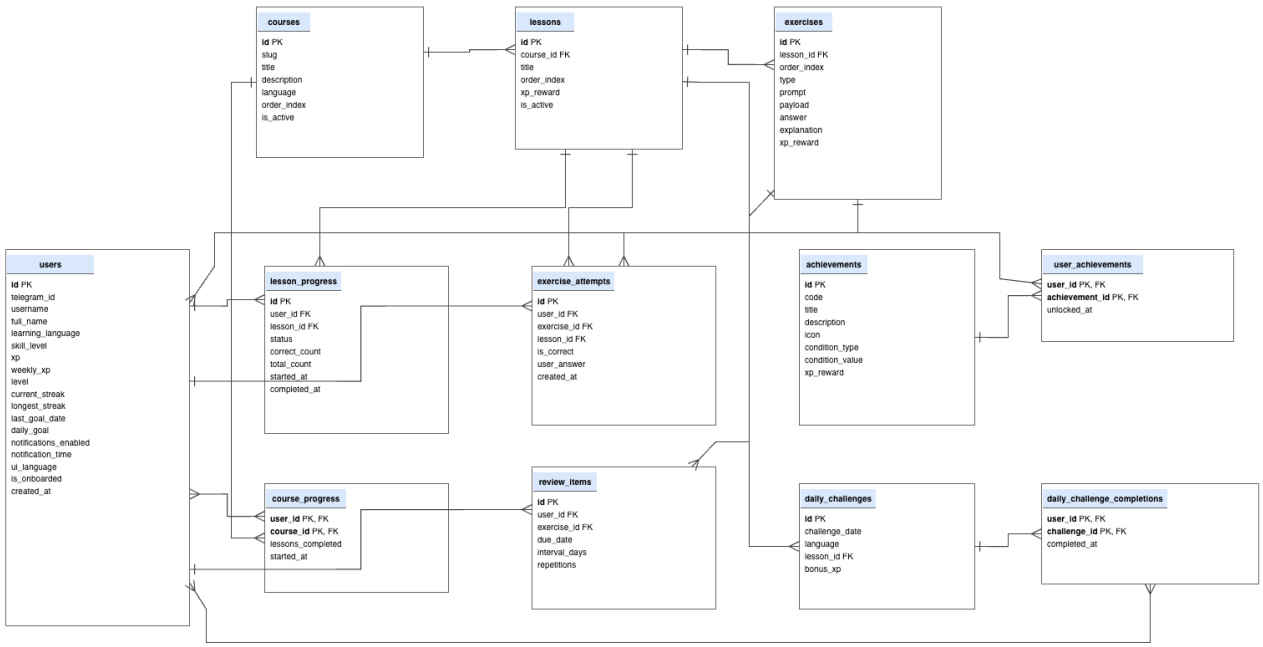

Рисунок 3.2 – ER-діаграма бази даних CodeQuest

Центральною є таблиця users, що зберігає обліковий запис користувача: ідентифікатор у Telegram, ім'я, обрану для вивчення мову програмування, рівень підготовки, накопичений досвід, досвід за поточний тиждень, рівень, поточну й найдовшу серію, дату останнього виконання щоденної цілі, розмір щоденної цілі, налаштування сповіщень і час нагадувань, мову інтерфейсу та ознаку завершення онбордингу. Такий склад дозволяє повноцінно описати стан користувача без потреби в додаткових таблицях.

Групу навчального контенту утворюють три таблиці. Таблиця courses зберігає курси з полями унікального текстового ідентифікатора, назви, опису, мови програмування та порядкового індексу. Таблиця lessons зберігає уроки, кожен з яких належить курсу через зовнішній ключ і має назву, порядковий індекс та винагороду досвіду. Таблиця exercises зберігає вправи: кожна вправа належить уроку, має тип, формулювання, а також два стовпці типу JSONB — вміст вправи та еталонну відповідь. Завдяки JSONB-стовпцям одна таблиця exercises обслуговує всі вісім типів вправ, попри відмінність їх внутрішньої структури.

Групу прогресу користувача утворюють чотири таблиці. Таблиця lesson_progress фіксує проходження уроків зі статусом, кількістю правильних та загальною кількістю вправ. Таблиця exercise_attempts зберігає кожну спробу виконання вправи з ознакою правильності й наданою відповіддю. Таблиця course_progress зберігає прогрес проходження курсів зі складеним первинним ключем з ідентифікаторів користувача й курсу. Таблиця review_items реалізує чергу інтервального повторення з полями дати наступного повторення, поточного інтервалу та кількості успішних повторень поспіль.

Групу гейміфікації утворюють таблиці achievements (каталог досягнень із кодом, назвою, описом, піктограмою, типом і значенням умови та винагородою), user_achievements (відкриті користувачами досягнення зі складеним ключем), daily_challenges (челенджі дня з унікальним обмеженням на пару «дата — мова») та daily_challenge_completions (виконані челенджі зі складеним ключем). Зв'язки «один до багатьох» реалізовано зовнішніми ключами з каскадним видаленням, а зв'язки «багато до багатьох» — таблицями зі складеними первинними ключами, що

гарантують унікальність зв'язку. Така схема повністю покриває потреби системи у зберіганні даних і водночас залишається простою та зрозумілою.

3.3. Реєстрація, онбординг та профіль користувача

Взаємодія користувача з ботом розпочинається з команди /start. Перед обробкою будь-якого оновлення проміжне програмне забезпечення автоматично знаходить користувача у базі даних за його ідентифікатором у Telegram, а якщо такого запису ще немає — створює новий. Завдяки цьому жоден обробник не потребує власної логіки реєстрації: на момент виклику обробника користувач уже гарантовано існує у базі та переданий у дані обробника.

Якщо користувач ще не пройшов онбординг, команда /start запускає процес початкового налаштування. Онбординг складається з трьох послідовних кроків, на кожному з яких бот ставить запитання й пропонує варіанти відповіді у вигляді вбудованої клавіатури. На першому кроці користувач обирає мову програмування для вивчення, на другому — рівень підготовки (початківець, середній або просунутий), на третьому — щоденну ціль у кількості уроків. Процес онбордингу подано на рисунку (див. рис. 3.3).

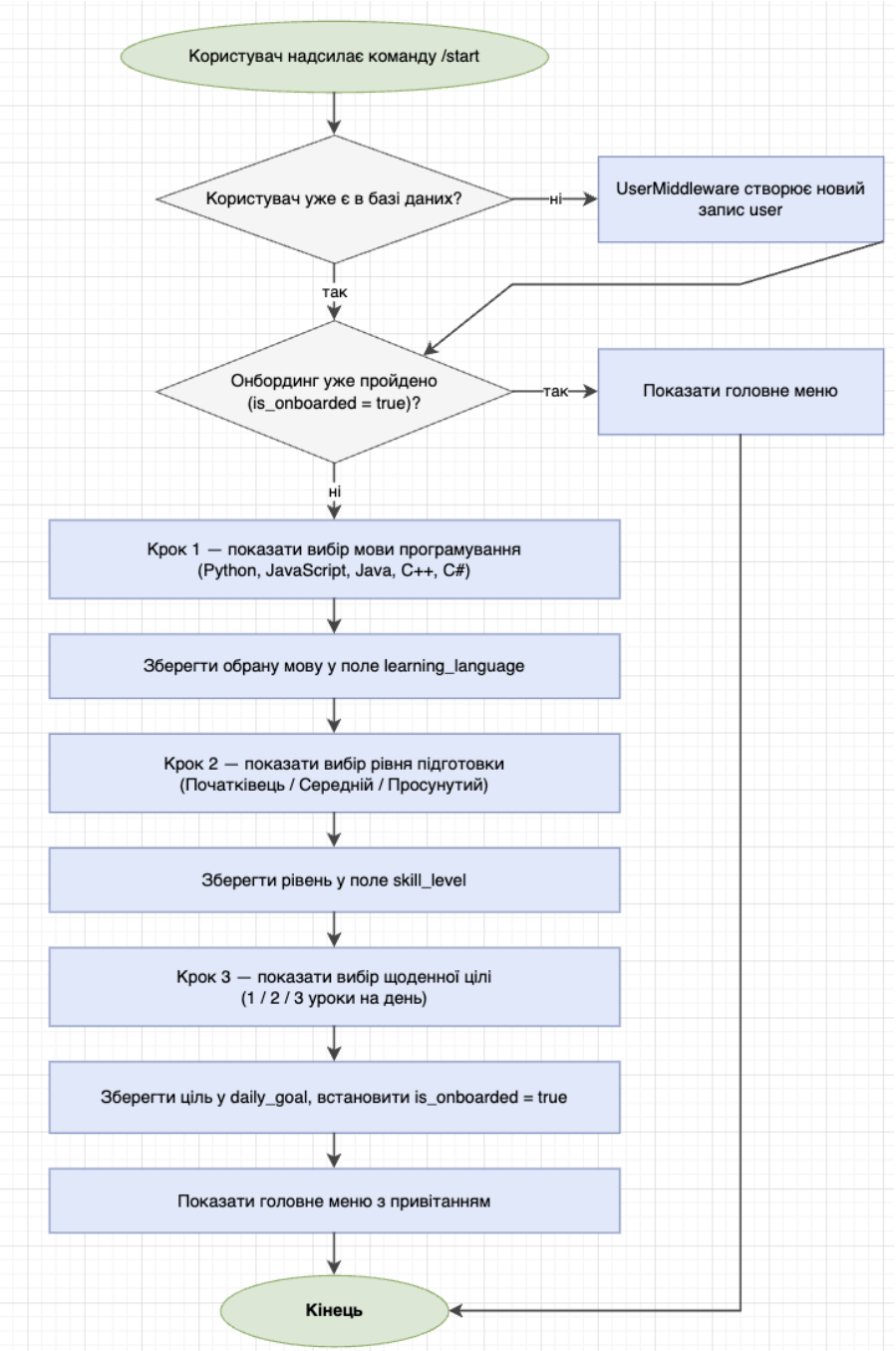

Рисунок 3.3 – Процес онбордингу нового користувача

Кожен крок онбордингу реалізовано окремим обробником, прив'язаним до фабрики callback-даних з певним значенням поля. Обробник зберігає обраний користувачем варіант у відповідне поле моделі користувача й переходить до наступного запитання. Після третього кроку ознаку завершення онбордингу встановлюють у позитивне значення, і користувач потрапляє до головного меню. Зміни до моделі користувача автоматично зберігаються у базі даних завдяки проміжному шару, що завершує транзакцію після успішної обробки оновлення.

Головне меню бота є основним вузлом навігації й подається повідомленням з вбудованою клавіатурою, кнопки якої ведуть до розділів «Навчання», «Челендж дня», «Повторення», «Рейтинг», «Профіль» та «Налаштування». Текст головного меню містить привітання, поточний рівень і звання користувача, накопичений досвід та поточну серію, що забезпечує миттєвий зворотний зв'язок про прогрес. Зовнішній вигляд головного меню наведено на рисунку (див. рис. 3.4).

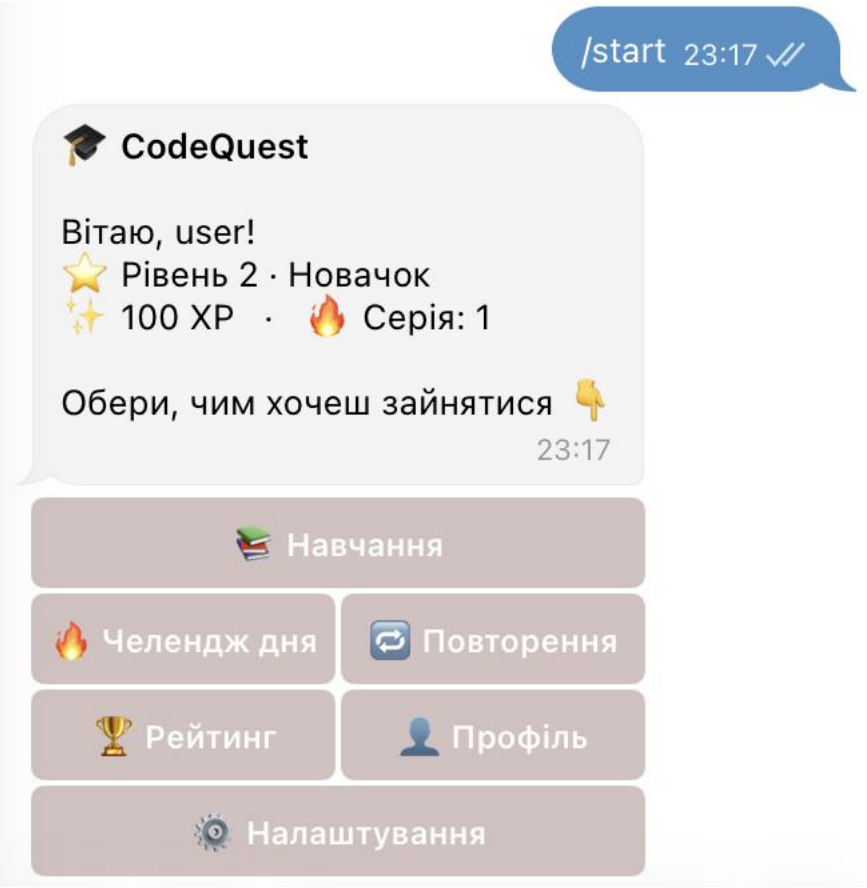

Рисунок 3.4 – Головне меню бота CodeQuest

Розділ профілю відображає докладну інформацію про користувача: ім'я, рівень і звання, смугу прогресу до наступного рівня, загальний досвід, поточну та рекордну серію, кількість пройдених уроків і точність відповідей. З профілю доступні підрозділи статистики, що показує прогрес по кожному курсу, та досягнень, де перелічено всі досягнення з позначкою відкритих і ще закритих. Підрозділ налаштувань дозволяє у будь-який момент змінити мову програмування, рівень, щоденну ціль, час нагадувань та увімкнути або вимкнути сповіщення.

3.4. Рушій уроків та реалізація восьми типів вправ

Рушій уроків є центральним компонентом системи. Урок — це послідовність із кількох інтерактивних вправ, проходження якої триває приблизно дві-три хвилини. Проходження уроку реалізовано як скінченний автомат: при старті уроку встановлюється стан активного проходження, а до даних стану записують перелік ідентифікаторів вправ, індекс поточної вправи, лічильник правильних відповідей, накопичений досвід та проміжний стан складних вправ. Дані стану зберігаються у Redis, тому проходження уроку коректно відновлюється між окремими натисканнями кнопок.

На кожному кроці рушій завантажує поточну вправу з бази даних, формує текст повідомлення та вбудовану клавіатуру відповідно до типу вправи й надсилає їх користувачу. Після відповіді користувача рушій перевіряє її, зіставляючи з еталоном, фіксує спробу у базі даних, передає результат у модуль інтервального повторення, оновлює лічильники у стані та показує користувачу зворотний зв'язок — позначку правильності й пояснення. Коли вправи в уроці вичерпано, рушій завершує урок, нараховує досвід і показує підсумок.

Система підтримує вісім типів інтерактивних вправ, кожен з яких має власну логіку відображення та перевірки. Зведення типів вправ разом зі структурою даних їх вмісту та еталонної відповіді наведено в таблиці 3.1.

Таблиця 3.1 — Типи інтерактивних вправ та структура їх даних

Тип вправи	**Призначення**	**Вміст вправи**	**Еталонна відповідь**
Картка теорії	пояснення з прикладом коду	текст, код	—
Тест із варіантами	обрати правильний варіант	варіанти	індекс правильного
Що виведе код	передбачити результат коду	код, варіанти	індекс правильного
Збери код	упорядкувати рядки	рядки	правильний порядок

	коду		
Встав пропуск	обрати пропущений токен	код, варіанти	індекс правильного
Знайди помилку	вказати хибний рядок	рядки коду	номер рядка
Правда чи хибність	оцінити твердження	твердження	істина або хибність
Зістав пари	зіставити елементи	лівий і правий списки	перелік пар

Логіку відображення та перевірки вправ зосереджено в окремому сервісному модулі. Функція відображення формує текст вправи залежно від її типу: для картки теорії додає пояснення та блок коду, для тесту й споріднених типів — код і варіанти, для вправи «Збери код» — поточний зібраний порядок рядків, для вправи «Зістав пари» — уже складені пари. Функція перевірки порівнює відповідь користувача з еталоном за правилом, специфічним для типу вправи: для вправ з вибором варіанта звіряються індекси, для вправи «Збери код» — повна послідовність, для вправи «Зістав пари» — множина пар без урахування порядку.

Особливістю реалізації є те, що складні вправи — «Збери код» та «Зістав пари» — виконуються в кілька натискань, тому потребують зберігання проміжного стану. Цей проміжний стан зберігається у даних FSM-стану і відображається користувачу після кожного натискання, доки вправу не буде завершено. Приклад проходження вправи типу «Збери код» наведено на рисунку (див. рис. 3.5).

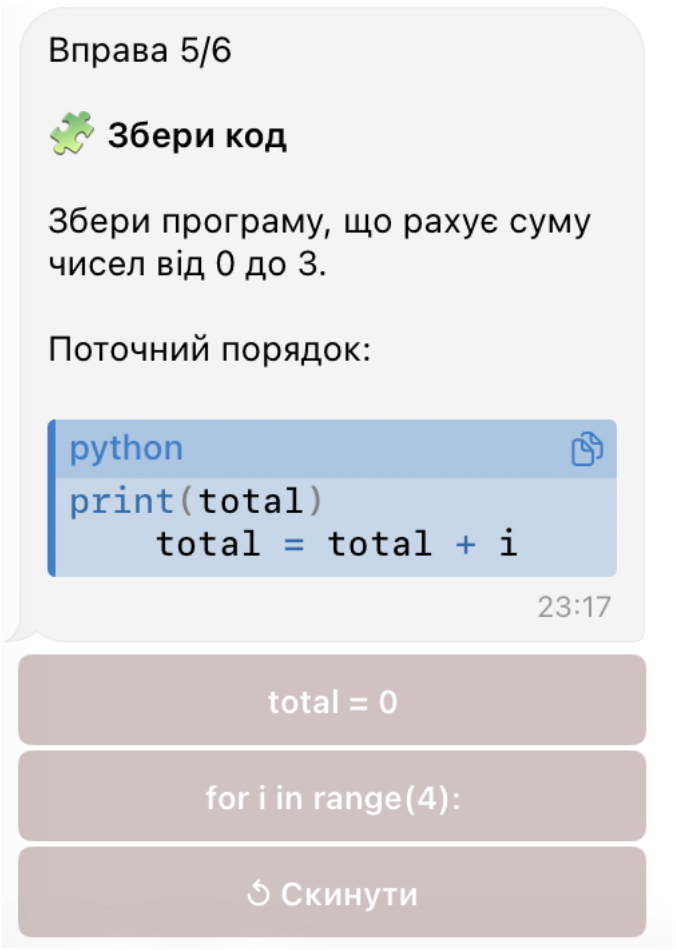

Рисунок 3.5 – Проходження вправи «Збери код»

Вбудовані клавіатури для вправ формуються окремим модулем-конструктором залежно від типу та поточного проміжного стану вправи. Для картки теорії це єдина кнопка «Далі», для тесту — кнопки з варіантами, для вправи «Знайди помилку» — кнопки з номерами рядків, для вправи «Збери код» — кнопки з ще не використаними рядками та кнопка скидання. Такий поділ між сервісом перевірки, конструктором клавіатур та обробниками подій забезпечує чистоту коду й полегшує додавання нових типів вправ.

3.5. Гейміфікація: досвід, рівні, звання та досягнення

Модуль гейміфікації реалізує механіки, що підтримують мотивацію користувача: бали досвіду, рівні, звання, досягнення та серію днів. Досвід нараховується за виконання вправ, завершення уроків, проходження челенджів дня та відкриття досягнень. При нарахуванні досвіду одночасно оновлюється і досвід за поточний тиждень, що використовується для тижневого рейтингу.

Рівень користувача визначається обсягом накопиченого досвіду за наперед заданою формулою: накопичений поріг рівня дорівнює добутку п'ятдесяти, номера рівня без одиниці та номера рівня. Це означає, що кожен наступний рівень потребує більше досвіду, ніж попередній, — поріг другого рівня становить сто одиниць досвіду, третього — триста, п'ятого — тисячу, десятого — чотири тисячі п'ятсот. Зростання порогів створює відчуття дедалі вагомішого досягнення з кожним рівнем. Рівень перераховується автоматично при кожній зміні досвіду, а факт переходу на новий рівень окремо повідомляється користувачу.

На основі рівня користувачу присвоюється звання, що є якісним маркером прогресу: рівні з першого до п'ятого відповідають званню «Новачок», з шостого до п'ятнадцятого — «Учень», з шістнадцятого до тридцятого — «Практик», з тридцять першого до п'ятдесятого — «Майстер», а починаючи з п'ятдесят першого — «Гуру». Звання відображаються у головному меню та профілі поряд із рівнем.

Система досягнень реалізована як набір умов, що автоматично перевіряються після завершення кожного уроку. Кожне досягнення має тип умови та порогове значення; підтримуються типи умов за кількістю пройдених уроків, кількістю бездоганних уроків, обсягом досвіду, найдовшою серією, кількістю виконаних челенджів та кількістю завершених курсів. Усього система містить тринадцять досягнень — від «Перші кроки» за перший пройдений урок до «Легенда CodeQuest» за три тисячі одиниць досвіду. Перевірка досягнень обчислює поточні показники користувача й порівнює їх з умовами ще не відкритих досягнень; за кожне

новоздобуте досягнення нараховується додатковий досвід, а користувач отримує сповіщення.

Серія днів обліковується окремим механізмом, прив'язаним до щоденної цілі. Після завершення уроку перевіряється, чи виконано щоденну ціль за поточний день. Якщо ціль уперше виконано сьогодні і попередній зарахований день був учора, серія збільшується на одиницю; якщо ж між днями стався розрив, серія починається заново з одиниці. Найдовша досягнута серія зберігається окремо як особистий рекорд користувача. Підсумковий екран уроку, що показує нарахований досвід, перехід на новий рівень, стан серії та нові досягнення, наведено на рисунку (див. рис. 3.6).

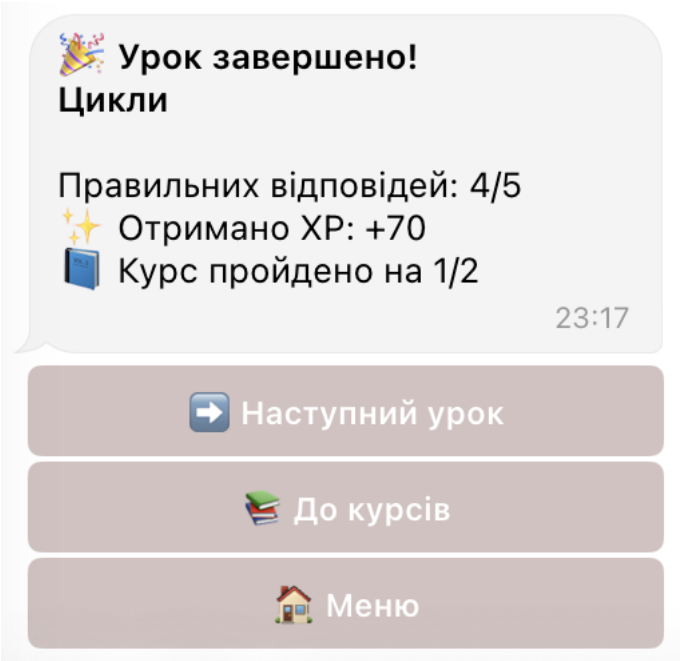

Рисунок 3.6 – Підсумковий екран уроку з нарахуванням досвіду

Сукупність описаних механік утворює цілісну систему позитивного підкріплення: кожна навчальна дія користувача дає видиму винагороду, а накопичений прогрес мотивує продовжувати навчання.

3.6. Щоденна ціль, челендж дня та рейтинг

Щоденна ціль — це обрана користувачем кількість уроків, які він планує проходити щодня; ціль може дорівнювати одному, двом або трьом урокам. Прогрес виконання цілі обчислюється підрахунком уроків, завершених за поточну добу, і відображається користувачу. Виконання щоденної цілі є умовою зарахування дня до серії, що пов'язує цей механізм із системою гейміфікації.

Челендж дня — це особливий урок, що пропонується користувачу один раз на день та винагороджується додатковим бонусом досвіду. Для кожної мови програмування генерується окремий челендж дня, що гарантується унікальним обмеженням бази даних на пару «дата — мова». Якщо челендж на поточну дату для обраної мови ще не існує, він створюється автоматично: серед усіх уроків відповідної мови випадково обирається один. Виконання челенджу фіксується в окремій таблиці, що унеможливлює повторне отримання бонусу того самого дня, та зараховується до відповідного досягнення.

Рейтинг забезпечує змагальний складник навчання й існує у двох різновидах. Глобальний рейтинг упорядковує користувачів за загальним накопиченим досвідом, тижневий — за досвідом, набраним протягом поточного тижня. Для обох рейтингів обчислюється не лише перелік лідерів, а й особиста позиція поточного користувача, навіть якщо він не входить до першої десятки. Тижневий рейтинг дає шанс відзначитися й новим користувачам, оскільки скидається щотижня. Зовнішній вигляд екрана рейтингу наведено на рисунку (див. рис. 3.7).

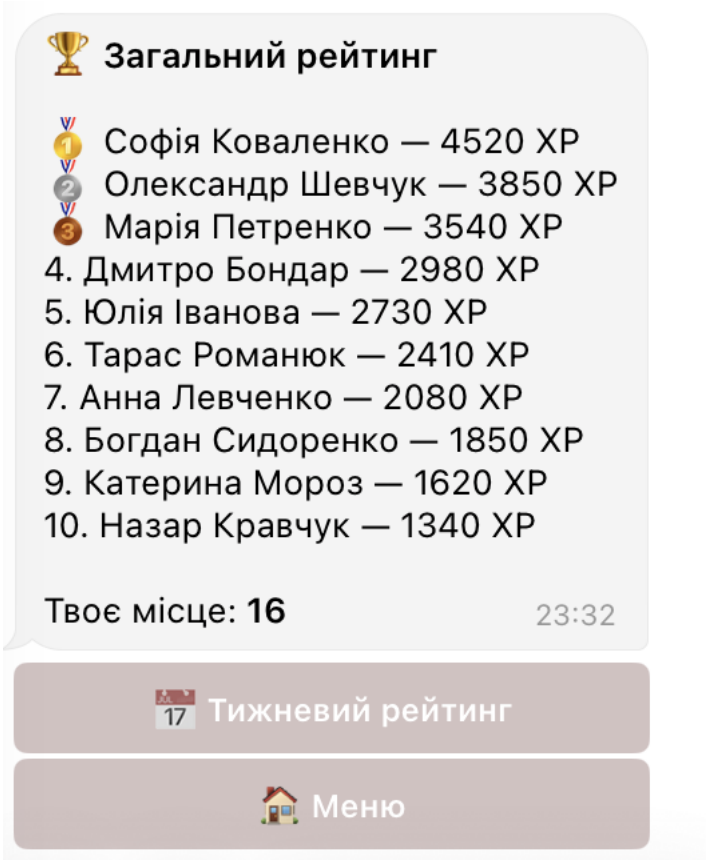

Рисунок 3.7 – Екран рейтингу гравців

Реалізацію рейтингу побудовано на ефективних запитах до бази даних: перелік лідерів отримується запитом з упорядкуванням за досвідом та обмеженням кількості, а позиція користувача — запитом, що підраховує кількість користувачів з більшим за його значенням досвідом. Це дозволяє швидко формувати рейтинг навіть за значної кількості користувачів, не завантажуючи всі записи до пам'яті застосунку.

3.7. Інтервальне повторення, планувальник та нагадування

Модуль інтервального повторення реалізує науково обґрунтований принцип розподіленого у часі закріплення матеріалу, описаний у підрозділі 2.4. Після перевірки кожної вправи її результат передається у цей модуль. Якщо відповідь була неправильною, вправа додається до черги повторення з найменшим інтервалом, а дата наступного повторення встановлюється на завтрашній день.

Якщо відповідь була правильною і вправа вже перебувала у черзі, її інтервал збільшується до наступного значення з послідовності.

Послідовність інтервалів повторення становить один, три, сім та чотирнадцять днів. Кожне успішне повторення переводить вправу до наступного інтервалу; після успішного проходження останнього інтервалу вправу вважають надійно закріпленою і вилучають із черги. Повторна помилка під час повторення повертає вправу до початкового інтервалу, забезпечуючи додаткове опрацювання проблемного матеріалу.

Користувач отримує доступ до черги повторення через окремий розділ «Повторення» головного меню. Розділ показує кількість вправ, готових до повторення на поточну дату, і дозволяє розпочати сесію повторення. Сесія повторення використовує той самий рушій уроків, що й звичайні уроки, але працює в окремому режимі: до неї добираються вправи з черги, термін повторення яких уже настав.

Для виконання періодичних задач, не пов'язаних з безпосередньою взаємодією користувача, у системі застосовано фоновий планувальник на основі бібліотеки APScheduler. Планувальник запускається разом із ботом і виконує три задачі за розкладом. Перша задача кожні тридцять хвилин розсилає нагадування користувачам, які увімкнули сповіщення, чий час нагадування настав і які ще не виконали щоденну ціль. Друга задача щодоби генерує челенджі дня для всіх мов програмування. Третя задача щопонеділка скидає тижневий рейтинг, обнуляючи тижневий досвід усіх користувачів [8].

Нагадування є важливим засобом формування навчальної звички. Текст нагадування коротко спонукає користувача приділити кілька хвилин уроку, не будучи нав'язливим. Завдяки тому, що нагадування надсилаються лише тим, хто ще не виконав щоденну ціль, користувач, який уже займався сьогодні, зайвого повідомлення не отримує. Час нагадувань користувач обирає самостійно у налаштуваннях, а за бажання може вимкнути сповіщення повністю.

3.8. Навчальний контент та розгортання системи

Навчальний контент — курси, уроки, вправи та досягнення — постачається у вигляді структурованих даних мовою Python, відокремлених від програмного коду. Кожна мова програмування має власний каталог, а кожен курс описано окремим файлом-модулем, що містить назву, опис та перелік уроків з вправами. Такий підхід робить контент легко розширюваним: щоб додати новий курс, достатньо створити один файл з даними, не вносячи жодних змін до коду застосунку.

При старті застосунку виконується наповнення бази даних контентом. Процедура наповнення є ідемпотентною: вона перевіряє, які курси вже наявні у базі за їх унікальними ідентифікаторами, і додає лише відсутні. Завдяки цьому повторні запуски бота не призводять до дублювання контенту, а додані до файлів нові курси автоматично з'являються у базі при наступному старті.

Станом на момент завершення роботи навчальний контент охоплює п'ять мов програмування — Python, JavaScript, Java, C++ та C#. Кожна мова має власний курс-трек приблизно однакового обсягу. Загалом контент налічує тридцять дев'ять курсів, дев'яносто уроків та п'ятсот сорок вправ, у яких задіяно всі вісім типів вправ. Теми курсів охоплюють основи й вивід даних, змінні та типи, умовні конструкції, цикли, функції, масиви та колекції, рядки й об'єктно-орієнтоване програмування; трек Python додатково містить рекурсію та алгоритми. Каталог курсів фільтрується за обраною користувачем мовою програмування.

Розгортання системи виконується засобами Docker та інструмента docker compose. Файл опису містить три сервіси: bot — застосунок бота, образ якого збирається з відповідного опису; postgres — реляційна СУБД на офіційному образі; redis — сховище станів на офіційному образі. Для бази даних і сховища передбачено постійні томи для збереження даних між перезапусками, а для бази даних — перевірку готовності, від якої залежить старт сервісу бота. Усі параметри передаються через файл середовища.

Для запуску системи достатньо створити файл середовища на основі наявного зразка, вписати в нього отриманий від службового бота @BotFather токен і виконати єдину команду запуску. Уся подальша ініціалізація — створення схеми

бази даних, наповнення контентом, запуск бота й планувальника — відбувається автоматично. Така простота розгортання робить систему придатною до запуску на будь-якій платформі, що підтримує Docker [10].

3.9. Інструкція для користувача

Робота з ботом розпочинається з його пошуку у Telegram за іменем користувача та надсилання команди /start. При першому запуску бот пропонує пройти онбординг: послідовно обрати мову програмування для вивчення, рівень підготовки та щоденну ціль. Усі відповіді надаються натисканням кнопок, тому налаштування займає кілька секунд. Після завершення онбордингу користувач потрапляє до головного меню.

Для початку навчання слід обрати у головному меню розділ «Навчання», далі — курс, а в ньому — урок. Уроки відкриваються послідовно: наступний урок стає доступним лише після завершення попереднього, що забезпечує поступове зростання складності. Під час уроку користувач виконує вправи натисканням кнопок: обирає варіант відповіді, упорядковує рядки коду, вставляє пропущений токен. Після кожної вправи бот показує, чи правильною була відповідь, та надає пояснення. Після останньої вправи відображається підсумок уроку з нарахованим досвідом.

Розділ «Челендж дня» пропонує особливий урок дня з бонусом досвіду; його можна пройти один раз на добу. Розділ «Повторення» містить вправи, у яких користувач раніше помилявся, і дозволяє закріпити складні теми. Розділ «Рейтинг» показує загальний і тижневий рейтинги гравців та особисту позицію користувача. Розділ «Профіль» відображає рівень, звання, досвід, серію, статистику й досягнення; його зовнішній вигляд наведено на рисунку (див. рис. 3.8).

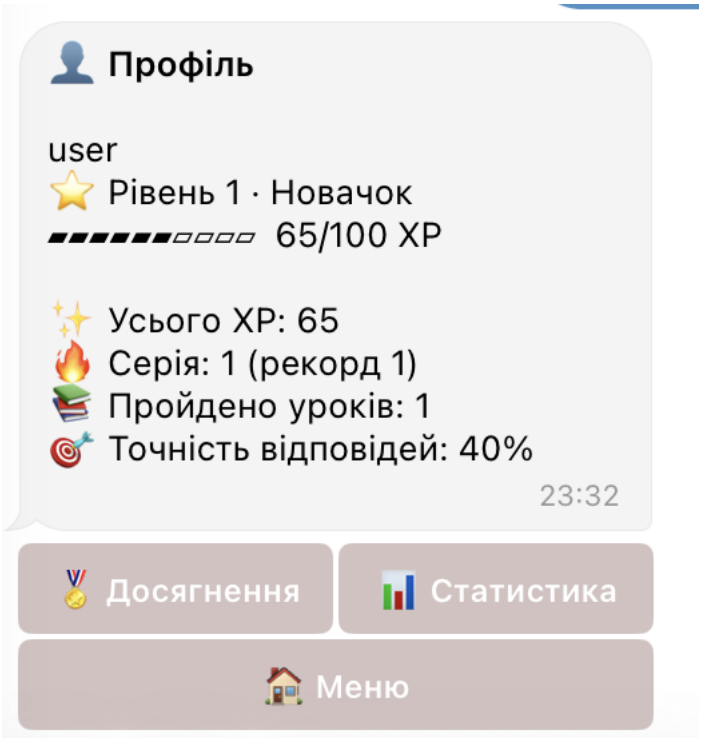

Рисунок 3.8 – Профіль користувача зі статистикою

У розділі «Налаштування» користувач може у будь-який момент змінити мову програмування, рівень підготовки, щоденну ціль, час нагадувань, а також увімкнути або вимкнути сповіщення. Для швидкої навігації передбачено команди: /menu відкриває головне меню, /help показує коротку довідку про можливості бота. Регулярне виконання щоденної цілі формує серію днів і відкриває нові досягнення, що є основним стимулом до системного навчання.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи поставлена мета — розробка Telegram-бота для вдосконалення навичок програмування, який знижує поріг входу до практики та формує звичку щоденних занять, — повністю досягнута. Розроблений програмний засіб «CodeQuest» є функціонально завершеним навчальним інструментом, готовим до використання.

У результаті виконання кваліфікаційної роботи отримано такі основні результати:

- проведено аналіз предметної області мікронавчання програмування, досліджено принципи гейміфікації, формування навчальної звички та інтервального повторення як основи ефективного навчального інструменту;
- виконано порівняльний аналіз існуючих рішень для навчання програмування, складено таблицю порівняння аналогів та обґрунтовано доцільність розробки власного Telegram-бота;
- досліджено теоретичні засади побудови Telegram-ботів: програмний інтерфейс Telegram Bot API, фреймворк aiogram, асинхронну модель виконання мови Python, реляційну модель даних та механізм скінченних автоматів;
- обґрунтовано вибір технологічного стеку — Python, aiogram, PostgreSQL, Redis, SQLAlchemy, APScheduler — та спроєктовано багатошарову архітектуру системи;
- спроєктовано реляційну схему бази даних з дванадцяти таблиць у третій нормальній формі для зберігання користувачів, навчального контенту, прогресу та гейміфікації;
- реалізовано модуль реєстрації та онбордингу з вибором мови програмування, рівня підготовки та щоденної цілі;
- реалізовано рушій уроків на основі скінченного автомата зі станом у Redis та підтримкою восьми типів інтерактивних вправ;

- реалізовано модуль гейміфікації: нарахування досвіду, рівні, звання, тринадцять досягнень та облік серії днів;
- реалізовано модулі щоденної цілі, челенджу дня та рейтингу — глобального і тижневого;
- реалізовано модуль інтервального повторення, що повертає помилкові вправи на повторне опрацювання за зростаючими інтервалами;
- реалізовано фоновий планувальник для нагадувань, генерації челенджів дня та скидання тижневого рейтингу;
- підготовлено навчальний контент для п'яти мов програмування — тридцять дев'ять курсів, дев'яносто уроків та п'ятсот сорок вправ — і конфігурацію для контейнерного розгортання.

Розроблений бот має ряд практичних переваг: повністю асинхронна архітектура забезпечує одночасне обслуговування багатьох користувачів; взаємодія виключно через екранні кнопки усуває фрикцію набору коду; модульна структура та відокремлений від коду контент спрощують розширення; контейнеризація робить розгортання простим і відтворюваним на будь-якій платформі. Усе це робить «CodeQuest» зручним інструментом для самостійного вивчення програмування.

Перспективи подальшого розвитку проєкту включають розширення навчального контенту новими курсами та мовами, додавання нових типів вправ, реалізацію режиму змагання двох користувачів у реальному часі, впровадження адаптивного добору вправ за рівнем користувача, додавання повної англомовної локалізації інтерфейсу та розробку адміністративної панелі для керування навчальним контентом.

Кваліфікаційна робота продемонструвала практичні навички системного аналізу, проєктування багатошарової архітектури, асинхронного програмування мовою Python, роботи з реляційними базами даних та сучасними практиками розгортання програмного забезпечення. Розроблений бот відповідає поставленим вимогам і готовий до використання за призначенням.

СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Python 3.12 Documentation. Python Software Foundation, 2024. Інтернет-доступ: https://docs.python.org/3/
2. aiogram 3 Documentation. Інтернет-доступ: https://docs.aiogram.dev/en/latest/
3. Telegram Bot API. Telegram Messenger, 2024. Інтернет-доступ: https://core.telegram.org/bots/api
4. SQLAlchemy 2.0 Documentation. Інтернет-доступ: https://docs.sqlalchemy.org/en/20/
5. PostgreSQL 16 Documentation. The PostgreSQL Global Development Group, 2023. Інтернет-доступ: https://www.postgresql.org/docs/16/index.html
6. asyncpg Documentation. Інтернет-доступ: https://magicstack.github.io/asyncpg/current/
7. Redis Documentation. Redis Ltd., 2024. Інтернет-доступ: https://redis.io/docs/latest/
8. APScheduler Documentation. Інтернет-доступ: https://apscheduler.readthedocs.io/en/3.x/
9. pydantic-settings Documentation. Інтернет-доступ: https://docs.pydantic.dev/latest/concepts/pydantic_settings/
10. Docker Compose specification. Docker Inc., 2024. Інтернет-доступ: https://docs.docker.com/compose/
11. Python asyncio — Asynchronous I/O. Python Software Foundation, 2024. Інтернет-доступ: https://docs.python.org/3/library/asyncio.html
12. PEP 8 — Style Guide for Python Code. Python Software Foundation, 2023. Інтернет-доступ: https://peps.python.org/pep-0008/
13. The Twelve-Factor App. Інтернет-доступ: https://12factor.net/
14. Ольховська О. В. Методичні рекомендації щодо виконання кваліфікаційної роботи здобувачами освітнього ступеня бакалавр спеціальності 122

«Комп'ютерні науки» / Олена Володимирівна Ольховська. – Полтава : ПУЕТ, 2024. – 67 с.

ДОДАТОК А

Файл config.py — завантаження конфігурації застосунку зі змінних середовища.

```
from pydantic_settings import BaseSettings, SettingsConfigDict


class Settings(BaseSettings):
model_config = SettingsConfigDict(
env_file=".env", env_file_encoding="utf-8", extra="ignore"
)

bot_token: str
postgres_host: str = "postgres"
postgres_port: int = 5432
postgres_user: str = "codequest"
postgres_password: str = "codequest"
postgres_db: str = "codequest"
redis_host: str = "redis"
redis_port: int = 6379
redis_db: int = 0

@property
def database_url(self) -> str:
return (
f"postgresql+asyncpg://{self.postgres_user}:{self.postgres_password}"
f"@{self.postgres_host}:{self.postgres_port}/{self.postgres_db}"
)

@property
def redis_url(self) -> str:
return f"redis://{self.redis_host}:{self.redis_port}/{self.redis_db}"
```

```
settings = Settings()
```

Файл models/base.py — базовий клас усіх ORM-моделей.

```
from datetime import datetime, timezone

from sqlalchemy.orm import DeclarativeBase


def utcnow() -> datetime:
return datetime.now(timezone.utc)


class Base(DeclarativeBase):
pass
```

Файл models/user.py — ORM-модель користувача.

```
from datetime import date, datetime, time
from typing import Optional

from sqlalchemy import BigInteger, Boolean, Date, DateTime, Integer,
String, Time
from sqlalchemy.orm import Mapped, mapped_column

from models.base import Base, utcnow


class User(Base):
__tablename__ = "users"

id: Mapped[int] = mapped_column(Integer, primary_key=True)
telegram_id: Mapped[int] = mapped_column(BigInteger, unique=True,
index=True)
username: Mapped[Optional[str]] = mapped_column(String(64))
full_name: Mapped[str] = mapped_column(String(255), default="")
```

```
learning_language: Mapped[str] = mapped_column(String(32),
default="python")
skill_level: Mapped[str] = mapped_column(String(16),
default="beginner")
xp: Mapped[int] = mapped_column(Integer, default=0)
weekly_xp: Mapped[int] = mapped_column(Integer, default=0)
level: Mapped[int] = mapped_column(Integer, default=1)
current_streak: Mapped[int] = mapped_column(Integer, default=0)
longest_streak: Mapped[int] = mapped_column(Integer, default=0)
last_goal_date: Mapped[Optional[date]] = mapped_column(Date)
daily_goal: Mapped[int] = mapped_column(Integer, default=1)
notifications_enabled: Mapped[bool] = mapped_column(Boolean,
default=True)
notification_time: Mapped[time] = mapped_column(Time, default=time(18,
0))
ui_language: Mapped[str] = mapped_column(String(8), default="uk")
is_onboarded: Mapped[bool] = mapped_column(Boolean, default=False)
created_at: Mapped[datetime] = mapped_column(DateTime(timezone=True),
default=utcnow)
```

Файл models/content.py — ORM-моделі навчального контенту: курс, урок, вправа.

```
from sqlalchemy import Boolean, ForeignKey, Integer, String, Text
from sqlalchemy.dialects.postgresql import JSONB
from sqlalchemy.orm import Mapped, mapped_column

from models.base import Base


class Course(Base):
__tablename__ = "courses"

id: Mapped[int] = mapped_column(Integer, primary_key=True)
slug: Mapped[str] = mapped_column(String(64), unique=True)
title: Mapped[str] = mapped_column(String(128))
```

```
description: Mapped[str] = mapped_column(Text, default="")
language: Mapped[str] = mapped_column(String(32), default="python")
order_index: Mapped[int] = mapped_column(Integer, default=0)
is_active: Mapped[bool] = mapped_column(Boolean, default=True)


class Lesson(Base):
__tablename__ = "lessons"

id: Mapped[int] = mapped_column(Integer, primary_key=True)
course_id: Mapped[int] = mapped_column(
ForeignKey("courses.id", ondelete="CASCADE"), index=True
)
title: Mapped[str] = mapped_column(String(128))
order_index: Mapped[int] = mapped_column(Integer, default=0)
xp_reward: Mapped[int] = mapped_column(Integer, default=20)
is_active: Mapped[bool] = mapped_column(Boolean, default=True)


class Exercise(Base):
__tablename__ = "exercises"

id: Mapped[int] = mapped_column(Integer, primary_key=True)
lesson_id: Mapped[int] = mapped_column(
ForeignKey("lessons.id", ondelete="CASCADE"), index=True
)
order_index: Mapped[int] = mapped_column(Integer, default=0)
type: Mapped[str] = mapped_column(String(16))
prompt: Mapped[str] = mapped_column(Text, default="")
payload: Mapped[dict] = mapped_column(JSONB, default=dict)
answer: Mapped[dict] = mapped_column(JSONB, default=dict)
explanation: Mapped[str] = mapped_column(Text, default="")
xp_reward: Mapped[int] = mapped_column(Integer, default=10)
```

Файл database/engine.py — створення асинхронного рушія та фабрики сесій бази даних.

```
from sqlalchemy.ext.asyncio import async_sessionmaker,
create_async_engine

from config import settings
from models import Base

engine = create_async_engine(settings.database_url,
pool_pre_ping=True)
sessionmaker = async_sessionmaker(engine, expire_on_commit=False)



async def init_models() -> None:
async with engine.begin() as conn:
await conn.run_sync(Base.metadata.create_all)
```

Файл bot.py — точка входу застосунку: ініціалізація бота, реєстрація проміжного програмного забезпечення, роутерів та планувальника.

```
import asyncio
import logging

from aiogram import Bot, Dispatcher
from aiogram.client.default import DefaultBotProperties
from aiogram.enums import ParseMode
from aiogram.fsm.storage.redis import RedisStorage
from aiogram.types import BotCommand

from config import settings
from database.engine import engine, init_models, sessionmaker
from handlers import courses, daily, errors, leaderboard, lesson,
profile, review
from handlers import settings as settings_handler
from handlers import start
from middlewares.database import DatabaseMiddleware
```

```
from middlewares.throttling import ThrottlingMiddleware
from middlewares.user import UserMiddleware
from scheduler.jobs import setup_scheduler
from seeds.seed import seed_content

logging.basicConfig(
level=logging.INFO,
format="%(asctime)s %(levelname)s %(name)s: %(message)s",
)
logger = logging.getLogger("codequest")

COMMANDS = [
BotCommand(command="menu", description="Головне меню"),
BotCommand(command="help", description="Довідка"),
BotCommand(command="start", description="Перезапустити бота"),
]


async def main():
await init_models()
async with sessionmaker() as session:
await seed_content(session)
await session.commit()

storage = RedisStorage.from_url(settings.redis_url)
bot = Bot(
token=settings.bot_token,
default=DefaultBotProperties(parse_mode=ParseMode.HTML),
)
dp = Dispatcher(storage=storage)

dp.update.outer_middleware(DatabaseMiddleware(sessionmaker))
dp.update.outer_middleware(UserMiddleware())
dp.message.middleware(ThrottlingMiddleware())
dp.callback_query.middleware(ThrottlingMiddleware())
```

```
for router in (
start.router,
courses.router,
lesson.router,
daily.router,
review.router,
leaderboard.router,
profile.router,
settings_handler.router,
errors.router,
):
dp.include_router(router)

scheduler = setup_scheduler(bot, sessionmaker)
scheduler.start()

await bot.set_my_commands(COMMANDS)
logger.info("CodeQuest запущено")
try:
await dp.start_polling(bot)
finally:
scheduler.shutdown(wait=False)
await bot.session.close()
await engine.dispose()


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

Файл middlewares/database.py — проміжний шар, що відкриває сесію бази даних для кожного оновлення.

```
from aiogram import BaseMiddleware


class DatabaseMiddleware(BaseMiddleware):
```

```
def __init__(self, sessionmaker):
self.sessionmaker = sessionmaker

async def __call__(self, handler, event, data):
async with self.sessionmaker() as session:
data["session"] = session
try:
result = await handler(event, data)
await session.commit()
return result
except Exception:
await session.rollback()
raise
```

Файл middlewares/user.py — проміжний шар автоматичної реєстрації користувача.

```
from aiogram import BaseMiddleware
from aiogram.types import Update

from repositories import users as users_repo

_USER_FIELDS = (
"message",
"edited_message",
"callback_query",
"inline_query",
"my_chat_member",
"chat_member",
"chat_join_request",
"poll_answer",
"pre_checkout_query",
)


def _from_user(update):
```

```
for field in _USER_FIELDS:
event = getattr(update, field, None)
if event is not None:
return getattr(event, "from_user", None)
return None


class UserMiddleware(BaseMiddleware):
async def __call__(self, handler, event, data):
session = data.get("session")
tg_user = _from_user(event) if isinstance(event, Update) else None
if session is not None and tg_user is not None and not tg_user.is_bot:
data["user"] = await users_repo.get_or_create(session, tg_user)
return await handler(event, data)
```

Файл repositories/content.py — функції доступу до навчального контенту.

```
from sqlalchemy import func, select

from models import Course, Exercise, Lesson


async def active_courses(session, language):
res = await session.execute(
select(Course)
.where(Course.is_active.is_(True), Course.language == language)
.order_by(Course.order_index)
)
return list(res.scalars().all())


async def get_course(session, course_id):
return await session.get(Course, course_id)


async def course_lessons(session, course_id):
```

```
res = await session.execute(
select(Lesson)
.where(Lesson.course_id == course_id, Lesson.is_active.is_(True))
.order_by(Lesson.order_index)
)
return list(res.scalars().all())


async def get_lesson(session, lesson_id):
return await session.get(Lesson, lesson_id)


async def lessons_for_language(session, language):
res = await session.execute(
select(Lesson)
.join(Course, Course.id == Lesson.course_id)
.where(Lesson.is_active.is_(True), Course.language == language)
)
return list(res.scalars().all())


async def lesson_exercises(session, lesson_id):
res = await session.execute(
select(Exercise)
.where(Exercise.lesson_id == lesson_id)
.order_by(Exercise.order_index)
)
return list(res.scalars().all())


async def get_exercise(session, exercise_id):
return await session.get(Exercise, exercise_id)


async def exercises_by_ids(session, ids):
if not ids:
```

```
return {}
res = await
session.execute(select(Exercise).where(Exercise.id.in_(ids)))
return {e.id: e for e in res.scalars().all()}


async def count_lessons(session, course_id):
res = await session.execute(
select(func.count())
.select_from(Lesson)
.where(Lesson.course_id == course_id, Lesson.is_active.is_(True))
)
return res.scalar_one()


async def has_courses(session):
res = await session.execute(select(Course.id).limit(1))
return res.first() is not None
```

Файл services/exercises.py — відображення та перевірка восьми типів вправ.

```
from utils.text import code_block, esc

SCORED_TYPES = {
"mcq",
"output",
"reorder",
"fill_gap",
"find_bug",
"true_false",
"match",
}

TYPE_TITLES = {
"theory": "□ Теорія",
"mcq": "□ Питання",
```

```
"output": "□ Що виведе код?",
"reorder": "□ Збери код",
"fill_gap": "□□ Встав пропуск",
"find_bug": "□ Знайди помилку",
"true_false": "□□ Правда чи хибність",
"match": "□ Зістав пари",
}


def is_scored(exercise) -> bool:
return exercise.type in SCORED_TYPES


def render(exercise, work=None) -> str:
work = work or {}
payload = exercise.payload or {}
parts = [f"<b>{esc(TYPE_TITLES.get(exercise.type, 'Вправа'))}</b>"]
if exercise.prompt:
parts.append(esc(exercise.prompt))

if exercise.type == "theory":
if payload.get("text"):
parts.append(esc(payload["text"]))
if payload.get("code"):
parts.append(code_block(payload["code"]))
elif exercise.type in ("mcq", "output", "fill_gap"):
if payload.get("code"):
parts.append(code_block(payload["code"]))
elif exercise.type == "find_bug":
lines = payload.get("lines", [])
numbered = "\n".join(f"{i + 1}  {line}" for i, line in
enumerate(lines))
parts.append(code_block(numbered))
elif exercise.type == "true_false":
if payload.get("statement"):
```

```
parts.append("«" + esc(payload["statement"]) + "»")
elif exercise.type == "reorder":
lines = payload.get("lines", [])
order = work.get("order", [])
if order:
assembled = "\n".join(lines[i] for i in order if 0 <= i < len(lines))
parts.append("Поточний порядок:")
parts.append(code_block(assembled))
else:
parts.append("<i>Натискай рядки у правильному порядку.</i>")
elif exercise.type == "match":
left = payload.get("left", [])
right = payload.get("right", [])
pairs = work.get("pairs", [])
if pairs:
shown = "\n".join(
f"• {esc(left[l])} → {esc(right[r])}"
for l, r in pairs
if 0 <= l < len(left) and 0 <= r < len(right)
)
parts.append("Складені пари:")
parts.append(shown)
pending = work.get("pending", -1)
if pending >= 0 and pending < len(left):
parts.append(f"Обрано зліва: <b>{esc(left[pending])}</b> — обери
пару.")
return "\n\n".join(parts)


def check(exercise, answer: dict) -> bool:
expected = exercise.answer or {}
kind = exercise.type
if kind in ("mcq", "output", "fill_gap"):
return answer.get("correct") == expected.get("correct")
if kind == "true_false":
return bool(answer.get("correct")) == bool(expected.get("correct"))
```

```
if kind == "find_bug":
return answer.get("line") == expected.get("line")
if kind == "reorder":
return list(answer.get("order", [])) == list(expected.get("order",
[]))
if kind == "match":
got = sorted(tuple(p) for p in answer.get("pairs", []))
want = sorted(tuple(p) for p in expected.get("pairs", []))
return got == want
return True


def correct_answer_text(exercise) -> str:
payload = exercise.payload or {}
expected = exercise.answer or {}
kind = exercise.type
if kind in ("mcq", "output", "fill_gap"):
options = payload.get("options", [])
index = expected.get("correct", 0)
if 0 <= index < len(options):
return str(options[index])
if kind == "true_false":
return "Правда" if expected.get("correct") else "Хибність"
if kind == "find_bug":
return f"рядок {expected.get('line', 0) + 1}"
if kind == "reorder":
lines = payload.get("lines", [])
order = expected.get("order", [])
return "\n".join(lines[i] for i in order if 0 <= i < len(lines))
if kind == "match":
left = payload.get("left", [])
right = payload.get("right", [])
return "\n".join(
f"• {left[l]} → {right[r]}"
for l, r in expected.get("pairs", [])
if 0 <= l < len(left) and 0 <= r < len(right)
```

```
)
return ""
```

Файл services/gamification.py — нарахування досвіду, перерахунок рівнів, серія та досягнення.

```
from datetime import date, timedelta

from repositories import content as content_repo
from repositories import daily as daily_repo
from repositories import gamification as gami_repo
from repositories import learning as learning_repo
from utils.text import level_for_xp


def add_xp(user, amount: int) -> bool:
if amount <= 0:
return False
before = user.level
user.xp += amount
user.weekly_xp += amount
user.level = level_for_xp(user.xp)
return user.level > before


async def update_streak(session, user) -> bool:
today = date.today()
if user.last_goal_date == today:
return False
done = await learning_repo.completed_on(session, user.id, today)
if done < user.daily_goal:
return False
if user.last_goal_date == today - timedelta(days=1):
user.current_streak += 1
else:
user.current_streak = 1
```

```
user.last_goal_date = today
if user.current_streak > user.longest_streak:
user.longest_streak = user.current_streak
return True


async def _completed_courses(session, user_id: int) -> int:
progress = await learning_repo.all_course_progress(session, user_id)
done = 0
for cp in progress:
total = await content_repo.count_lessons(session, cp.course_id)
if total > 0 and cp.lessons_completed >= total:
done += 1
return done


async def check_achievements(session, user) -> list:
achievements = await gami_repo.all_achievements(session)
if not achievements:
return []
unlocked = await gami_repo.unlocked_ids(session, user.id)
pending = [a for a in achievements if a.id not in unlocked]
if not pending:
return []
stats = {
"lessons": await learning_repo.count_completed(session, user.id),
"perfect": await learning_repo.count_perfect(session, user.id),
"xp": user.xp,
"streak": user.longest_streak,
"challenges": await daily_repo.count_completed(session, user.id),
"courses": await _completed_courses(session, user.id),
}
newly = []
for achievement in pending:
if stats.get(achievement.condition_type, 0) >=
achievement.condition_value:
```

```
await gami_repo.unlock(session, user.id, achievement.id)
add_xp(user, achievement.xp_reward)
newly.append(achievement)
return newly
```

Файл services/spaced_repetition.py — алгоритм інтервального повторення.

```
from datetime import date, timedelta

from models import ReviewItem
from repositories import learning as learning_repo

INTERVALS = [1, 3, 7, 14]


def _due(repetition: int) -> tuple[int, int]:
repetition = max(0, min(repetition, len(INTERVALS) - 1))
return repetition, INTERVALS[repetition]


async def register_result(session, user_id: int, exercise_id: int,
is_correct: bool):
item = await learning_repo.get_review_item(session, user_id,
exercise_id)
today = date.today()

if is_correct:
if item is None:
return
if item.repetitions + 1 >= len(INTERVALS):
await learning_repo.delete_review_item(session, item)
return
repetition, interval = _due(item.repetitions + 1)
item.repetitions = repetition
item.interval_days = interval
item.due_date = today + timedelta(days=interval)
```

```
return

repetition, interval = _due(0)
if item is None:
session.add(
ReviewItem(
user_id=user_id,
exercise_id=exercise_id,
repetitions=repetition,
interval_days=interval,
due_date=today + timedelta(days=interval),
)
)
else:
item.repetitions = repetition
item.interval_days = interval
item.due_date = today + timedelta(days=interval)
```

Файл keyboards/exercises.py — конструктори вбудованих клавіатур для вправ.

```
from aiogram.types import InlineKeyboardMarkup
from aiogram.utils.keyboard import InlineKeyboardBuilder

from callbacks import Ex
from utils.text import trim


def exercise_kb(exercise, work: dict) -> InlineKeyboardMarkup:
b = InlineKeyboardBuilder()
kind = exercise.type
payload = exercise.payload or {}

if kind == "theory":
b.button(text="Далі □□", callback_data=Ex(action="next", value=0))
b.adjust(1)
return b.as_markup()
```

```
if kind in ("mcq", "output", "fill_gap"):
for index, option in enumerate(payload.get("options", [])):
b.button(text=trim(option, 56), callback_data=Ex(action="ans",
value=index))
b.adjust(1)
return b.as_markup()

if kind == "true_false":
b.button(text="□ Правда", callback_data=Ex(action="ans", value=1))
b.button(text="□ Хибність", callback_data=Ex(action="ans", value=0))
b.adjust(2)
return b.as_markup()

if kind == "find_bug":
for index in range(len(payload.get("lines", []))):
b.button(text=f"Рядок {index + 1}", callback_data=Ex(action="ans",
value=index))
b.adjust(3)
return b.as_markup()

if kind == "reorder":
order = work.get("order", [])
for index, line in enumerate(payload.get("lines", [])):
if index not in order:
b.button(text=trim(line, 56), callback_data=Ex(action="ord",
value=index))
b.button(text="↺ Скинути", callback_data=Ex(action="reset", value=0))
b.adjust(1)
return b.as_markup()

if kind == "match":
pairs = work.get("pairs", [])
pending = work.get("pending", -1)
used_left = {l for l, _ in pairs}
used_right = {r for _, r in pairs}
```

```
for index, item in enumerate(payload.get("left", [])):
if index not in used_left:
mark = "□ " if index == pending else ""
b.button(text=mark + trim(item, 38), callback_data=Ex(action="ml",
value=index))
for index, item in enumerate(payload.get("right", [])):
if index not in used_right:
b.button(text=trim(item, 38), callback_data=Ex(action="mr",
value=index))
b.button(text="↺ Скинути", callback_data=Ex(action="reset", value=0))
b.adjust(1)
return b.as_markup()

b.button(text="Далі □□", callback_data=Ex(action="next", value=0))
b.adjust(1)
return b.as_markup()


def feedback_kb(is_last: bool) -> InlineKeyboardMarkup:
b = InlineKeyboardBuilder()
text = "□ Завершити урок" if is_last else "Далі □□"
b.button(text=text, callback_data=Ex(action="next", value=0))
b.adjust(1)
return b.as_markup()
```

Файл utils/text.py — допоміжні функції форматування та обчислення рівнів.

```
import html

from aiogram.exceptions import TelegramBadRequest

RANKS = (
(5, "Новачок"),
(15, "Учень"),
(30, "Практик"),
(50, "Майстер"),
```

```
)


def esc(text) -> str:
return html.escape(str(text))


def code_block(text) -> str:
return f"<pre><code>{html.escape(str(text))}</code></pre>"


def inline_code(text) -> str:
return f"<code>{html.escape(str(text))}</code>"


def xp_for_level(level: int) -> int:
return 50 * (level - 1) * level


def level_for_xp(xp: int) -> int:
level = 1
while xp_for_level(level + 1) <= xp:
level += 1
return level


def rank_name(level: int) -> str:
for threshold, name in RANKS:
if level <= threshold:
return name
return "Гуру"


def level_progress(xp: int):
level = level_for_xp(xp)
current = xp_for_level(level)
```

```
nxt = xp_for_level(level + 1)
return level, xp - current, nxt - current


def progress_bar(done: int, total: int, width: int = 10) -> str:
filled = round(width * done / total) if total > 0 else 0
filled = max(0, min(width, filled))
return "▰" * filled + "▱" * (width - filled)


def plural(n: int, one: str, few: str, many: str) -> str:
n = abs(n)
if n % 10 == 1 and n % 100 != 11:
return one
if 2 <= n % 10 <= 4 and not 12 <= n % 100 <= 14:
return few
return many


def trim(text, limit: int = 40) -> str:
text = " ".join(str(text).split())
if len(text) <= limit:
return text
return text[: limit - 1] + "…"


async def safe_edit(message, text, reply_markup=None):
try:
await message.edit_text(text, reply_markup=reply_markup)
except TelegramBadRequest:
try:
await message.answer(text, reply_markup=reply_markup)
except TelegramBadRequest:
pass
```

Файл scheduler/jobs.py — фонові періодичні задачі планувальника.

```
import logging
from datetime import date, datetime, timezone

from apscheduler.schedulers.asyncio import AsyncIOScheduler
from apscheduler.triggers.cron import CronTrigger

from keyboards.inline import LANGUAGES
from repositories import gamification as gami_repo
from repositories import learning as learning_repo
from repositories import users as users_repo
from services import daily as daily_service

logger = logging.getLogger("codequest")


def setup_scheduler(bot, sessionmaker) -> AsyncIOScheduler:
scheduler = AsyncIOScheduler(timezone="UTC")
scheduler.add_job(
_send_reminders, CronTrigger(minute="*/30"), args=(bot, sessionmaker)
)
scheduler.add_job(
_create_daily_challenge, CronTrigger(hour=0, minute=5),
args=(sessionmaker,)
)
scheduler.add_job(
_reset_weekly,
CronTrigger(day_of_week="mon", hour=0, minute=1),
args=(sessionmaker,),
)
return scheduler


async def _send_reminders(bot, sessionmaker):
now = datetime.now(timezone.utc)
```

```
today = date.today()
async with sessionmaker() as session:
users = await users_repo.all_users(session)
for user in users:
if not user.notifications_enabled or not user.is_onboarded:
continue
moment = user.notification_time
if moment.hour != now.hour:
continue
if not now.minute <= moment.minute < now.minute + 30:
continue
async with sessionmaker() as session:
done = await learning_repo.completed_on(session, user.id, today)
if done >= user.daily_goal:
continue
try:
await bot.send_message(
user.telegram_id,
"□ Час для уроку CodeQuest! Кілька хвилин — і ти ближче "
"до мети. Натисни /menu □",
)
except Exception as exc:
logger.warning("Не вдалося надіслати нагадування: %s", exc)


async def _create_daily_challenge(sessionmaker):
async with sessionmaker() as session:
for code, _ in LANGUAGES:
await daily_service.get_or_create_today(session, code)
await session.commit()


async def _reset_weekly(sessionmaker):
async with sessionmaker() as session:
await gami_repo.reset_weekly(session)
await session.commit()
```

```
logger.info("Тижневий рейтинг скинуто")
```