ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Цифровий сервіс доступу до персоналізованого розкладу занять та сповіщень на основі Telegram-бота

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення освітньо-професійної програми «Технології цифрового розвитку»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Богдан СИМЧУК
(підпис)

Виконав: здобувач вищої освіти групи ТЦР-44

Богдан СИМЧУК

Керівник: Ірина ГНІДЕНКО
ст.викладач

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку Ступінь

вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технологій цифрового розвитку
_____Вікторія ЖЕБКА
« _____» _____2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Симчуку Богдану Васильовичу

1. Тема кваліфікаційної роботи: «Цифровий сервіс доступу до персоналізованого розкладу занять та сповіщень на основі Telegram-бота»
керівник кваліфікаційної роботи д-р техн. наук, професор Ірина ЗАМРІЙ, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «_____» березня 2026 року № ____.

2. Строк подання кваліфікаційної роботи «___» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: Вихідні дані до кваліфікаційної роботи: теоретичні відомості про архітектуру програмних інтерфейсів (API) та скінченні автомати (FSM), опис методів асинхронного програмування, синтаксичного аналізу текстового вводу та проєктування реляційних баз даних, технічна документація Telegram Bot API, мови програмування Python та супутніх бібліотек (aiogram/telebot, asyncio, psycopg2.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз існуючих методів та технологій розробки діалогових інтерфейсів в контексті задачі автоматизації доступу до навчального розкладу.
2. Проєктування застосунку (Telegram-бота) для автоматизованого управління розкладом занять.
3. Програмна реалізація та опис функціонування застосунку (Telegram-бота) для

автоматизованого управління розкладом занять.

4. Тестування застосунку.
5. Перелік графічного матеріалу: *презентація*
   Аналіз аналогів.
   Вимоги до програмного забезпечення.
   Програмні засоби реалізації.
   Діаграма діяльності.
   Екранні форми.
   Апробація результатів дослідження
6. Дата видачі завдання «20» лютий 2026 року.

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	21.02.-04.03.2026	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2026	
3	Аналіз застосунку для інтерактивного навчання методам шифрування з елементами гейміфікації	14.03-21.03.2026	
4	Проектування навчального модулю з елементами гейміфікації для вивчення методів шифрування	22.03-09.04.2026	
5	Реалізація навчального модуля з елементами гейміфікації для вивчення методів шифрування	10.04-28.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2026	
8	Розробка демонстраційних матеріалів	12.05-18.05.2026	
9	Попередній захист роботи	19.05-30.05.2026	

Здобувач(ка) вищої освіти _____ Богдан СИМЧУК
(підпис)

Керівник
кваліфікаційної роботи _____ Ірина ГНІДЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 71 стор., 3 рис., 12 джерел.

Мета роботи – підвищення ефективності, оперативності та зручності доступу учасників освітнього процесу до навчального розкладу шляхом розробки автоматизованого Telegram-бота.

Об’єкт дослідження – процес інформаційного забезпечення та управління навчальним розкладом у закладах вищої освіти.

Предмет дослідження – програмно-алгоритмічні засоби та архітектурні рішення для реалізації автоматизованої системи (чат-бота) доступу до навчального розкладу.

Короткий зміст роботи: У кваліфікаційній роботі проаналізовано проблематику традиційних підходів до поширення навчального розкладу та досліджено існуючі програмні рішення (LMS-системи, нативні мобільні застосунки, цифрові календарі). На основі аналізу було спроєктовано та програмно реалізовано автоматизовану систему у вигляді Telegram-бота з використанням мови програмування Python та реляційної бази даних PostgreSQL для надійного збереження даних.

Програмно реалізовано модулі синтаксичного аналізу (парсингу) та валідації текстового вводу за допомогою регулярних виразів, а також підсистему управління діалогами на базі архітектури скінченного автомата (FSM). Впроваджено механізми асинхронної обробки запитів для забезпечення високої продуктивності та стійкості системи під час пікових ранкових навантажень. Реалізовано функціонал безпечного додавання, видалення та перегляду розкладу на день чи тиждень, а також алгоритм відстеження поточної та наступної навчальної дисципліни в режимі реального часу.

КЛЮЧОВІ СЛОВА: TELEGRAM-БОТ, РОЗКЛАД ЗАНЯТЬ, PYTHON, POSTGRESQL, АВТОМАТИЗАЦІЯ ОСВІТНЬОГО ПРОЦЕСУ, АСИНХРОННА ОБРОБКА, СКІНЧЕННИЙ АВТОМАТ (FSM).

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) – прикладний програмний інтерфейс для взаємодії між різними програмними компонентами.

FSM (Finite State Machine) – скінченний автомат, модель поведінки бота для керування станами діалогу.

IDE (Integrated Development Environment) – інтегроване середовище розробки програмного забезпечення.

JSON (JavaScript Object Notation) – текстовий формат обміну даними, що базується на парах «ключ-значення».

JSON Schema – мова опису структури та правил валідації JSON-даних.

PaaS (Platform as a Service) – модель хмарних обчислень, що надає платформу для розгортання додатків.

SDK (Software Development Kit) – набір інструментів розробника для створення програмного забезпечення.

UI (User Interface) – інтерфейс користувача, засоби взаємодії людини з програмою.

URL (Uniform Resource Locator) – уніфікований покажчик ресурсу в мережі.

ПЗ – програмне забезпечення.

БД – база даних.

ТЗ – технічне завдання.

ВСТУП

Використання ботів у месенджері Telegram сьогодні є одним із найбільш дієвих способів автоматизації рутинних процесів та налагодження якісного зв'язку з аудиторією. Такі програмні рішення успішно функціонують у бізнес-середовищі, освітній галузі та сфері розваг. Нижче ми розглянемо засадничі принципи створення ботів, їхню типологію та моделі поведінки користувачів.

Критерії розробки функціонального бота

Для того, щоб програмний продукт був ефективним, розробнику слід зосередитися на таких пунктах:

1. Технологічний стек. Важливо правильно обрати Telegram Bot API, мову програмування (наприклад, Python чи Node.js) та відповідний інструментарій, як-от PyTelegramBotAPI.

2. Безпека середовища. Обов'язковою умовою є конфіденційність даних, що забезпечується протоколами SSL та TLS, а також коректним поводженням з індивідуальними tokens.

3. Користувацький інтерфейс та UX. Логіка спілкування має бути простою, а вхідні повідомлення – легкими для сприйняття.

4. Стабільність. Програма має опрацьовувати вхідні запити в режимі real-time без критичних затримок.

5. Контроль якості. Перед запуском бот проходить ретельне тестування для виявлення та усунення помилок.

Залежно від мети та способу реалізації, боти поділяються на кілька категорій:

За напрямом діяльності: – Інформаційні (трансляція новин чи прогнозів). – Сервісні (допомога у покупках чи нагадування). – Розважальні (ігрові сценарії та вікторини).

За методом управління: – Текстові, що реагують на конкретні commands. – Навігаційні, де основою є інтерактивне menu та кнопки. – Розумні боти, що імітують

природну мову завдяки AI.

За рівнем реалізації: від елементарних скриптів до розгалужених систем, інтегрованих із зовнішніми web-services.

Особливості Use Case діаграм

Діаграма прецедентів допомагає візуалізувати всі сценарії, за якими User може взаємодіяти з програмою. Вона відображає архітектуру дій та реакцій системи:

Актор (користувач) надсилає запит через інтерфейс Telegram. – Етап authorization відкриває доступ до особистих налаштувань. – Ввід commands запускає виконання певного алгоритму. – Отримання даних або завершення цільової дії (наприклад, бронювання) стає логічним фіналом сесії.

Ці аспекти складають теоретичну базу, необхідну для проєктування сучасних інтелектуальних помічників, забезпечуючи їхню корисність для кінцевого споживача.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ЗАСОБІВ РЕАЛІЗАЦІЇ

1.1 Актуальність автоматизації доступу до навчального розкладу

Зараз важко знайти університет, де не намагалися б впровадити якісь цифрові фішки, бо це реально спрощує життя. Але як не крути, а основою всього навчання залишається звичайний розклад. Це ніби головний орієнтир і для нас, студентів, і для викладачів. Якщо кожен точно знає, де і коли буде пара, то й весь освітній процес іде як треба. Проблема лише в тому, як саме ми дізнаємося про цей графік.

Старі підходи, коли треба було бігти до роздрукованого аркуша на стіні або шукати на сайті якийсь PDF чи таблицю XLSX, сьогодні вже зовсім не годяться. Головна біда таких форматів – їхня статичність. Університетське життя постійно підкидає сюрпризи: хтось із викладачів захворів, якусь аудиторію зачинили на ремонт, або взагалі пари раптово перенесли в онлайн через безпеку. У таких випадках завантажені файли просто перетворюються на сміття, бо вони не оновлюються автоматично. Студенти приходять не туди, марнують час, і це лише додає хаосу. Постійно перевіряти сторінку сайту вручну теж задоволення сумнівне.

До того ж, дивитися величезні електронні таблиці з телефона – це окремий вид муки. На невеликому екрані доводиться постійно щось масштабувати, рухати вправо-вліво, намагаючись знайти свій факультет та групу серед сотень інших. Ми зараз майже все робимо через смартфони, тому й розклад має бути під них підлаштований. Інформація повинна з'являтися перед очима миттєво, а не змушувати користувача працювати «фільтром» для купи зайвих даних.

Були спроби робити окремі мобільні застосунки для кожного ВНЗ, але це виявилося не дуже практично. По-перше, розробка та підтримка під iOS чи Android коштує купу грошей і часу. По-друге, самі студенти часто не хочуть ставити зайвий софт, який займає пам'ять і вимагає постійної авторизації. Набагато простіше використовувати те, що вже є під рукою.

Саме тому зараз найкращим варіантом виглядають системи на базі

месенджерів, зокрема в Telegram. Це середовище, де молодь проводить левову частку свого часу. Інтегрувати розклад туди – це логічний крок, бо не треба нічого додатково встановлювати. Такі боти працюють швидко, не вантажать телефон і, що найголовніше, дозволяють надсилати миттєві сповіщення. Якщо в розкладі щось змінилося за п'ять хвилин до пари, студент одразу отримає про це сигнал.

Сама робота з ботом побудована дуже просто. Користувачеві не треба нічого шукати вручну – натиснув кнопку на клавіатурі типу Inline чи Reply, і система миттєво видала персоналізований список занять саме для його групи. Все відформатовано так, щоб було зручно читати з ходу. Це прибирає зайвий інформаційний шум і реально економить час.

Тож створення такого Telegram-бота для доступу до навчальних планів – це не просто експеримент, а дуже корисна і актуальна штука. Це допомагає налагодити нормальний зв'язок між закладом та студентами, робить навчання більш організованим і дає впевненість, що в тебе під рукою завжди свіжа інформація. Саме тому для своєї розробки я обрав зв'язку мови Python та середовища PyCharm, як найбільш оптимальний набір інструментів для такого завдання.

1.2 Аналіз інтерфейсів взаємодії: переваги месенджера Telegram

Під час розробки автоматизованих систем для інформування дуже важливо вгадати з форматом, у якому користувачі будуть взаємодіяти з програмою. Якщо подивитися на те, куди рухається сучасна розробка, то стає помітно, що люди все частіше обирають не окремі сайти чи додатки, а формат Conversational UI. У цьому плані месенджер Telegram став справжньою екосистемою, яка ідеально підходить для запуску сервісних ботів. Він має кілька серйозних переваг, які виділяють його серед конкурентів.

Перш за все, варто сказати про технологічну відкритість Bot API. Якщо порівнювати з тим же WhatsApp чи Viber, де все набагато складніше в налаштуванні, то тут розробник отримує повну свободу дій. Можна використовувати Inline-кнопки

або Reply-клавіатури, що дозволяє зробити інтерфейс зрозумілим навіть без інструкцій. Це дуже зручно, бо студенту не треба нічого друкувати вручну – достатньо просто натиснути на кнопку, і розклад уже на екрані. Такий підхід майже виключає помилки при введенні даних.

Ще один сильний аргумент – це повна кросплатформеність. Бот однаково добре працює і на смартфоні з iOS чи Android, і на ноутбуці. Для студентського життя це важливо, бо зранку розклад дивляться з телефона, а вже в університеті – з комп'ютера. Оскільки всі чати зберігаються в хмарі, історія повідомлень завжди під рукою, незалежно від того, яким пристроєм ви користуєтеся в цей момент.

З точки зору користувача, такий формат взагалі не викликає стресу. Майже кожен студент і так щодня користується месенджерами, тому розібратися з ботом можна за секунду. Не потрібно нічого додатково завантажувати чи створювати нові акаунти. Бот просто висить у списку діалогів поруч із особистим листуванням, що робить перевірку розкладу звичайною справою, яка не потребує зайвих зусиль.

Також не можна ігнорувати швидкість і те, наскільки мало ресурсів споживає такий сервіс. Текстові дані передаються миттєво навіть тоді, коли інтернет працює дуже слабко. А система автоматичних сповіщень дозволяє оперативно повідомити всіх про зміни чи скасування пари. Повідомлення просто вискакує на головному екрані смартфона, і його важко пропустити.

Питання безпеки тут теж вирішено грамотно. Використання протоколу MTProto забезпечує стабільність зв’язку та захищає дані від стороннього втручання. Хоча розклад не є таємницею, надійність роботи сервісу все одно стоїть на першому місці.

Для розробника ж головна вигода в тому, що не треба проходити довгі перевірки в магазинах додатків на кшталт App Store чи Google Play. Ви просто запускаєте бота, і він працює. Це дозволяє зосередитися на головному – на якості даних та логіці самої програми. У підсумку, такий вибір платформи є оптимальним, бо він поєднує в собі швидкість, зручність та потужний функціонал для цифровізації навчання.

1.3 Огляд існуючих систем управління розкладом та чат-ботів

Зараз на ринку софту для університетів вистачає різних інструментів, що допомагають якось впорядкувати розклад. Але щоб зрозуміти, чому ми зупинилися саме на месенджері, варто розібратися, як працюють інші системи та в чому їхні головні слабкості. Загалом усі наявні рішення можна розділити на великі навчальні платформи, окремі мобільні додатки, хмарні календарі та чат-боти.

Перша і найбільш масова група – це гігантські системи на кшталт Moodle чи Canvas, або ж внутрішні університетські портали. Там розклад зазвичай інтегрований із журналом та особистим кабінетом, що ніби зручно. Але з точки зору простого студента все виглядає складніше. Щоб просто глянути наступну пару, треба відкрити браузер, ввести логін та пароль, іноді пройти двофакторну перевірку, а потім ще й шукати потрібний пункт у заплутаному меню. Це занадто довгий шлях для такої швидкої задачі. До того ж, часто такі сайти погано адаптовані під екрани смартфонів, що тільки додає дискомфорту.

Другий варіант – це повноцінні додатки під iOS та Android. Вони працюють швидко, можуть зберігати дані без інтернету та надсилати системні повідомлення. Проте їх розробка коштує дуже дорого, бо треба створювати окремі версії під різні системи. Самі студенти теж не дуже люблять забивати пам'ять телефона програмою, яка потрібна лише для однієї функції. Досвід показує, що такі додатки часто видаляють одразу після сесії, щоб звільнити місце.

Третій підхід базується на використанні хмарних сервісів через протокол iCalendar. Користувач просто підписується на посилання, і пари з'являються в його Google Calendar чи Apple Calendar. Це непогано, бо розклад завжди в органайзері, але тут є свої підводні камені. Налаштувати це правильно може не кожен, а головне – синхронізація часто затримується на кілька годин або навіть на добу. Якщо розклад змінився в останній момент, студент про це просто не дізнається вчасно. Та й зворотного зв'язку тут немає: ви не можете запитати в календаря розклад іншої групи чи викладача.

Четвертий сегмент, який зараз активно розвивається – це спеціальні боти в месенджерах. Багато провідних українських вишів вже мають такі системи, створені або офіційно, або активними студентами. Технічно це виглядає як зв'язка клієнта та сервера, де Telegram чи Viber виступають лише оболонкою, а вся логіка обробки запитів та отримання даних з бази лежить на сервері.

Вивчення таких продуктів показує, що боти мають купу переваг. Будь-які оновлення в коді програми миттєво стають доступними для всіх без необхідності щось перекачувати. Завдяки кнопкам на зразок Inline чи Reply keyboards навігація стає настільки простою, що навіть не треба нічого писати – просто тиснеш на потрібний варіант. Також боти дозволяють робити масові розсилки, що критично важливо при термінових оголошеннях. І, звісно, вони працюють на будь-якому пристрої, де є месенджер.

Втім, аналіз показує, що багато нинішніх ботів мають суттєві недоліки. Найчастіше це аматорські проєкти, які просто «падають» вранці, коли тисячі студентів одночасно намагаються глянути розклад. Інша біда – використання нестабільних методів збору даних із сайтів, які ламаються при будь-якій зміні дизайну сторінки університету.

Отже, можна зробити висновок, що саме Telegram-бот є найбільш перспективним шляхом, бо він обходить обмеження важких систем та дорогих додатків. Але щоб такий сервіс працював надійно, при розробці треба приділити особливу увагу архітектурі сервера. Потрібно правильно налаштувати роботу зі станами сесій через state-machine, передбачити обробку помилок та алгоритми кешування даних, щоб система залишалася стабільною за будь-яких умов

1.4 Обґрунтування вибору технологічного стеку

Вибір конкретних інструментів для розробки – це завжди питання відповідальності, адже від цього залежить не тільки швидкість роботи, а й те, наскільки зручно буде підтримувати систему в майбутньому. Для створення нашого бота ми зупинилися на мові Python, яка зараз є одним із найкращих варіантів для таких завдань. Вона

дозволяє писати чистий код і значно прискорює весь процес розробки, що особливо важливо на етапі створення прототипу.

Важливо, що в роботі ми використовуємо об'єктно-орієнтований підхід. Це дозволяє нам описати всі ключові елементи системи, як-от стан чату чи навчальну дисципліну, у вигляді окремих класів. Такий підхід робить структуру програми логічною та допомагає уникнути плутанини в майбутньому, коли проєкт почне розростатися. Ми можемо чітко розмежувати, де зберігаються дані, а де прописана логіка їхньої обробки.

Окремо варто зупинитися на асинхронності. Наш бот – це програма, яка постійно обмінюється даними з серверами Telegram та базою даних. Уявіть ситуацію, коли вранці сотні студентів одночасно хочуть перевірити розклад перед парами. Якби ми писали код звичайним синхронним методом, бот би просто почав «зависати», намагаючись відповісти кожному по черзі. Використання бібліотеки aiogram та модуля asyncio дозволяє серверу працювати з багатьма запитами одночасно. Програма не чекає відповіді від одного користувача, а моментально перемикається на обробку іншого повідомлення, що забезпечує дуже швидкий відгук.

Щоб бот розумів, на якому саме етапі діалогу перебуває людина, ми впровадили механізм скінченних автоматів, або FSM. Це допомагає програмі «пам’ятати» контекст розмови – наприклад, який курс чи групу вибрав студент крок тому. Без цього бот просто не зміг би вести послідовний діалог і помилявся б при натисканні на кнопки. Кожен крок вибору чітко фіксується в системі, що робить інтерфейс бота надійним.

Для збереження всієї інформації про користувачів та самі графіки занять ми обрали базу даних SQLite. Це дуже легке і зручне рішення, яке не потребує налаштування окремого сервера на етапі запуску, але при цьому гарантує порядок у даних. Якщо ж у майбутньому аудиторія бота виросте до десятків тисяч користувачів, архітектура дозволить нам легко перейти на більш потужну систему, наприклад PostgreSQL. У підсумку такий набір технологій дозволяє отримати стабільний та продуктивний сервіс, який буде зручним для кожного студента.

1.4.1 Мова програмування Python та її переваги для розробки чат-ботів

Сьогодні у сфері розробки програмного забезпечення застосовується величезна кількість інструментів, проте саме Python здобула особливе визнання завдяки своїй лаконічності та інтуїтивній зрозумілості. Цю об’єктно-орієнтовану мову високого рівня, яку Гвідо ван Россум представив світу ще у 1991 році, часто рекомендують як ідеальний вибір для початківців. Водночас вона залишається надзвичайно потужним засобом для реалізації масштабних та складних рішень. Попри те, що з моменту її створення минуло понад тридцять років і її можна вважати відносно старою, Python стабільно утримує високі позиції, посідаючи четверту сходинку в авторитетному рейтингу TIOBE. Такий успіх цілком закономірний, адже синтаксис цієї мови вирізняється надзвичайною простотою та елегантністю. На відміну від багатьох інших мов, тут не використовуються фігурні дужки для розділення логічних блоків. Замість цього застосовується система відступів (табуляція), що суттєво зменшує обсяг написаного коду та робить його читання й написання максимально комфортним. Додатковою перевагою є висока портативність: розроблені скрипти без проблем запускаються на різноманітних операційних платформах, серед яких Windows, Linux, UNIX, Amigo та Mac OS. Важливо й те, що розробнику не доводиться вручну турбуватися про управління пам'яттю чи збір сміття, як це відбувається у випадку з C або C++.

Показовим є той факт, що цю технологію активно впроваджують у свої продукти такі світові гіганти, як Google, відеохостинг YouTube, космічне агентство NASA, а також компанії Cisco, Intel, iRobot та відома файлообмінна мережа BitTorrent. Усі ці перелічені характеристики формують образ Python як потужного, швидкого та портативного інструменту з відкритим вихідним кодом, що легко підтримує інші технології.

Для забезпечення ефективної багатозадачності у програмах часто використовується механізм багатопотоковості. Завдяки йому програма здатна виконувати більше ніж одну справу одночасно. Хоча технічно різні потоки можуть не виконуватися абсолютно паралельно в одну й ту саму мілісекунду, ця парадигма

створює повноцінний ефект одночасної роботи. У контексті мого бота це має безпосереднє практичне застосування: з одного боку, програма безперервно підтримує діалог з користувачем та обробляє його запити, а з іншого – у фоновому режимі самостійно перевіряє базу даних на наявність днів народження серед збережених контактів, щоб своєчасно вивести відповідне сповіщення.

З технічного погляду, потік – це незалежна лінія виконання всередині загального процесу, яку система може планувати автономно від іншого коду. Для того щоб почати працювати з цим інструментом, необхідно звернутися до стандартної бібліотеки Python, яка містить необхідний набір класів. Основним серед них є клас Thread. Щоб ініціювати новий паралельний процес, достатньо створити екземпляр цього класу та запустити його за допомогою методу .start(). Простими словами, це певна послідовність програмних інструкцій, яка здатна працювати ізольовано.

Ще одним критично важливим поняттям є демони. Це специфічні процеси, які функціонують виключно у фоновому режимі, і потоки відіграють для них ключову роль. У середовищі Python потокові демони мають одну унікальну властивість: вони автоматично зупиняються відразу після завершення роботи основної програми. Якщо ж у системі виконуються звичайні потоки, які не є демонами, програма буде чекати на повне завершення їхньої роботи, перш ніж остаточно закритися.

У розробці також активно застосовується JSON (JavaScript Object Notation) – відкритий формат текстового обміну даними, який історично виник на базі мови JavaScript. Попри цей тісний зв'язок у назві, сьогодні JSON є абсолютно незалежним і надзвичайно простим у використанні інструментом. Найчастіше його застосовують під час створення вебзастосунків. Зокрема, він є невіддільною частиною систем, що працюють із запитами Ajax (Asynchronous JavaScript And XML). Це особливий підхід до проєктування користувацьких інтерфейсів, який дозволяє завантажувати нові дані з сервера непомітно для користувача, вже після того, як вебсторінка була завантажена.

Такі механізми дають змогу оновлювати інформацію на екрані без повного перезавантаження сторінки або ж непомітно відправляти запити на сервер у

фоновому режимі. Звісно, найчастіше формат JSON згадується саме в контексті веб-розробки, проте його сфера застосування набагато ширша. Цей стандарт ідеально підходить для будь-яких завдань, де виникає потреба зберегти структуровані дані у вигляді звичайного тексту.

Головним аргументом на користь JSON є його феноменальна простота: структура документа залишається інтуїтивно зрозумілою як для автоматичних парсерів, так і для людини, яка читає код. По суті, весь файл складається з логічних пар «ключ-значення», що згруповані всередині об’єктів. Крім того, синтаксичні правила цього формату базуються на логіці, яка добре знайома кожному, хто працював із такими мовами програмування, як Python, C, C++, Java, PHP або JavaScript.

Якщо порівнювати зберігання інформації у файлах JSON із розгортанням повноцінної зовнішньої бази даних, можна помітити певні закономірності. Хоча класичні бази є більш потужними та функціональними, вони вимагають написання додаткових складних запитів, налаштування з'єднань та виділення серверних ресурсів, що суттєво уповільнює процес збереження та отримання інформації в невеликих проєктах.

Саме з цих міркувань я вирішив використати JSON як локальну базу даних для свого застосунку. Архітектура мого сховища побудована за дуже простою та ефективною моделлю: головним ідентифікатором (ключем) виступає унікальний номер користувача, а відповідним значенням є комплексний об’єкт, що містить абсолютно всю необхідну інформацію про цю особу.

Для реалізації програмної логіки були залучені спеціалізовані бібліотеки та API:

1. Першим і найголовнішим інструментом стала бібліотека python-telegram-bot (або просто PTB). Це потужне рішення, яке забезпечує чистий асинхронний інтерфейс для взаємодії з Telegram Bot API та є повністю сумісним з версіями Python 3.7 і вище. Окрім базових методів для роботи з мережею, цей пакет пропонує набір високорівневих інструментів, зосереджених у підмодулі telegram.ext, що суттєво полегшують та пришвидшують розробку. Бібліотека вирізняється

сучасним підходом та зрозумілою архітектурою. Її творці написали чудову документацію, а спільнота завжди готова допомогти у спеціальних чатах підтримки. У межах мого проєкту абсолютно всі команди бота були реалізовані з використанням функціоналу саме цієї бібліотеки.

2. Другим важливим компонентом став модуль Datetime. Це один із найважливіших стандартних інструментів Python, який гарантує максимальну гнучкість при роботі з часовими відмітками. Хоча деякі його аспекти можуть здатися специфічними на перший погляд, загалом його логіка перегукується з підходами інших мов. У цій кваліфікаційній роботі Datetime застосовується для формування запитів, пов'язаних із днями народження, а також для правильного форматування дат і перевірки їхньої валідності.

3. Третім елементом є Schedules – максимально простий та легкий API для планування фонових завдань. Ця бібліотека не має жодних зовнішніх залежностей та може похвалитися ідеальним тестовим покриттям. Звісно, вона не претендує на звання універсального планувальника для складних корпоративних систем, проте бездоганно справляється з базовими задачами. Я імплементував її спеціально для налаштування системи автоматичних нагадувань про дні народження.

Основою архітектури написаного коду є об'єктно-орієнтоване програмування (ООП). Це фундаментальна парадигма розробки, яка будується навколо концепції створення об'єктів. З технічного погляду, об'єкти є конкретними екземплярами класів, які об'єднують у собі стан (дані, що зберігаються у вигляді полів, атрибутів чи властивостей) та поведінку (програмний код, реалізований через процедури або методи). Використання ООП допомагає розбити складний монолітний код на логічні, незалежні блоки, які можна багаторазово використовувати в різних частинах застосунку.

До фундаментальних принципів об'єктно-орієнтованого підходу належать наступні концепції:

1. Класи та об'єкти. Клас можна уявити як своєрідне креслення або базовий шаблон, за яким конструюються програмні сутності. Своєю чергою, об'єкт є вже готовим, конкретним екземпляром цього шаблону, який наповнений реальними

даними та здатний виконувати закладені в класі функції.

2. Інкапсуляція. Цей принцип дозволяє надійно упакувати змінні та методи, що з ними працюють, в єдину захищену оболонку. Завдяки інкапсуляції внутрішня механіка роботи об'єкта залишається прихованою від зовнішнього середовища, що надійно захищає дані від випадкового чи несанкціонованого втручання з інших частин програми.

3. Наслідування. Цей механізм надає можливість новому класу (який прийнято називати підкласом) переймати всі властивості та методи вже існуючого базового класу. Такий підхід стимулює повторне використання написаного коду та дозволяє будувати стрункі ієрархічні структури без зайвого дублювання.

4. Четвертим важливим елементом є поліморфізмЦей принцип дає змогу об'єктам, які належать до різних класів, по-різному реагувати на виклик одних і тих самих методів. Завдяки цьому програма стає гнучкішою та універсальнішою у використанні.

5. Наступна концепція – це абстракція. Вона допомагає приховати зайві технічні деталі та показати складні системи у вигляді спрощених моделей. Іншим програмістам стає набагато простіше працювати з кодом, оскільки вони бачать лише публічний інтерфейс і не повинні вникати у внутрішні процеси об'єкта.

Загалом використання об'єктно-орієнтованого підходу робить програми надійнішими та простішими в обслуговуванні. Зменшується залежність між окремими компонентами, що дуже допомагає при масштабуванні проєкту. До речі, цю парадигму підтримує багато популярних мов, зокрема і C++.

Щодо середовища розробки, то для написання коду я обрав PyCharm. Це спеціалізоване інтегроване середовище (або IDE), створене компанією JetBrains спеціально для мови Python. Програма має безліч корисних функцій для написання, тестування та відлагодження коду. Наприклад, вона вміє автоматично вирішувати проблеми з імпортами та легко інтегрується з різними системами керування версіями.

Цікаво поглянути на історію цього продукту. Компанія JetBrains розпочала роботу над ним приблизно у 2009 році, коли популярність Python почала стрімко

зростати у світі програмування. Команда розробників уже мала солідний досвід, адже раніше вони створили відоме середовище IntelliJ IDEA для мови Java. Перша повноцінна версія побачила світ у 2010 році і майже одразу здобула прихильність серед розробників завдяки інтуїтивному інтерфейсу та широким можливостям.

З часом програма активно розвивалася, отримувала підтримку нових фреймворків та інструментів для підвищення продуктивності. Сьогодні це практично незамінна річ як для новачків, які лише вчаться програмувати, так і для професіоналів.

Популярність PyCharm зумовлена низкою серйозних переваг. По-перше, тут дуже зручно редагувати код завдяки автодоповненню, миттєвій перевірці синтаксису та можливостям рефакторингу. По-друге, середовище чудово працює з віртуальними оточеннями на кшталт Virtualenv чи Conda, тому керувати залежностями проєкту максимально просто. Окремо варто виділити інструменти для відлагодження. Наявність точок зупинки та зручний перегляд значень змінних економлять купу часу при пошуку помилок.

Крім того, PyCharm надає вбудовану підтримку популярних бібліотек та фреймворків, таких як Django, Flask, Pandas або NumPy. Не виникає жодних проблем і з системами контролю версій на кшталт Git чи SVN, що дозволяє ефективно працювати над кодом у команді. Також передбачена зручна інтеграція з базами даних та серверами. А оскільки програма належить до великого сімейства продуктів JetBrains, її логіка буде знайома тим, хто раніше працював у їхніх редакторах для інших мов.

Звісно, ідеальних інструментів не існує, і це середовище теж має певні мінуси. Найбільший недолік – це високі вимоги до апаратних ресурсів. На слабких комп'ютерах програма може відчутно гальмувати, особливо якщо обробляється велика кількість файлів. До того ж далеко не весь функціонал є безкоштовним. Базова версія Community Edition доступна всім, але для повноцінної роботи з веб-фреймворками доведеться купувати платну Professional Edition.

Для декого інтерфейс здається занадто перевантаженим функціями, і новачкам буває важко одразу розібратися у всіх налаштуваннях. Також іноді виникають певні

труднощі при налаштуванні підключення до віддалених серверів, особливо якщо вони знаходяться за специфічними мережами.

Якщо порівнювати PyCharm з іншими популярними редакторами, то ситуація виглядає наступним чином. Наприклад, Visual Studio Code (або VS Code) – це скоріше легкий текстовий редактор, який можна налаштувати під Python за допомогою розширень. Він споживає значно менше ресурсів комп'ютера і є більш гнучким, тоді як продукт від JetBrains – це вже готове повноцінне середовище.

Інший аналог – Jupyter Notebook. Цей інструмент працює інакше: він дозволяє виконувати код невеликими блоками і одразу бачити результат. Його найчастіше використовують для аналізу даних та складання звітів, а не для написання класичних програм. А ось середовище Spyder більше орієнтоване на наукові обчислення та має специфічні інструменти для візуалізації масивів даних. Кожен із цих інструментів має свою аудиторію, проте для розробки великих проєктів зазвичай обирають саме PyCharm.

Отже, можна зробити висновок, що PyCharm є справді потужним інструментом для розробки на мові Python. Його багатий функціонал, надійність та інтеграція з сучасними технологіями перекривають такі недоліки, як високе споживання оперативної пам'яті чи наявність платних версій. Вибір завжди залежить від конкретних завдань, але для створення ботів чи складних програм це одне з найкращих рішень на ринку.

Популярність середовища PyCharm серед розробників зумовлена цілим рядом вагомих переваг. Насамперед варто відзначити надзвичайно зручні інструменти для роботи з кодом. Програма не просто підсвічує синтаксис, а пропонує розумне автодоповнення, допомагає швидко проводити рефакторинг та автоматично виправляє проблеми з імпортами. Це значно прискорює рутинну роботу програміста.

Також це середовище чудово ладнає з віртуальними оточеннями. Завдяки вбудованій інтеграції з такими інструментами, як Virtualenv або Conda, керувати залежностями в проєкті стає набагато простіше. Окремої уваги заслуговує система відлагодження коду. Замість того, щоб шукати помилки наосліп, розробник може ставити точки зупинки, переглядати поточні значення змінних та аналізувати стек

викликів безпосередньо в інтерфейсі програми.

Ще один великий плюс – це нативна підтримка безлічі популярних бібліотек та фреймворків. Робота з Django, Flask, SQLAlchemy, NumPy чи Pandas стає значно зручнішою, що неабияк допомагає при створенні вебзастосунків чи програм для аналітики. Не менш важливою є можливість зручної спільної роботи. Середовище без проблем працює з популярними системами контролю версій на кшталт Git, Mercurial або SVN. Це дозволяє команді ефективно вести спільну розробку та слідкувати за всіма змінами в коді.

До додаткових зручностей можна віднести можливість підключення до зовнішніх ресурсів, таких як бази даних чи віддалені сервери, прямо з робочого вікна. А оскільки PyCharm створений компанією JetBrains, він має спільну логіку інтерфейсу з іншими їхніми продуктами, наприклад з IntelliJ IDEA. Це суттєво полегшує життя тим, хто пише код кількома різними мовами програмування. Загалом усі ці особливості роблять програму дуже потужним інструментом, що підвищує якість коду та швидкість його написання.

Однак, незважаючи на всі ці плюси, програма має і свої слабкі сторони. Найбільш відчутним недоліком є висока вимогливість до ресурсів комп'ютера. Через свою багатофункціональність середовище може досить сильно навантажувати систему, що стає справжньою проблемою для власників слабких ПК або при роботі над дуже великими проєктами.

Інший нюанс стосується фінансової сторони. Звісно, розробники пропонують безкоштовну версію під назвою Community Edition, але її можливості обмежені. Якщо вам знадобиться повноцінна підтримка вебфреймворків на кшталт Django чи Flask, доведеться купувати платну підписку на Professional Edition.

Крім того, інтерфейс програми може здатися занадто перевантаженим для новачків. Величезна кількість кнопок, меню та налаштувань часто лякає тих, хто тільки починає свій шлях у програмуванні, і на освоєння всіх можливостей середовища потрібен час. Також іноді виникають складнощі при спробі налаштувати стабільне з'єднання з віддаленими серверами, особливо якщо вони сховані за специфічними мережевими конфігураціями. Нарешті, користувачі іноді скаржаться

на певне падіння продуктивності самого редактора, коли доводиться обробляти гігантські масиви даних або відкривати проєкти з тисячами файлів.

Щоб краще зрозуміти місце PyCharm на ринку, варто порівняти його з іншими популярними інструментами для розробки на мові Python. Наприклад, часто виникає вибір між ним та Visual Studio Code (або коротко VS Code). Головна різниця полягає в тому, що продукт від JetBrains – це одразу готове до роботи монолітне середовище з усім необхідним функціоналом. Натомість VS Code є просто легким текстовим редактором, який споживає мінімум ресурсів і налаштовується під конкретні потреби виключно за рахунок встановлення додаткових плагінів.

Інший цікавий аналог – це Jupyter Notebook. Тут підходи до написання коду кардинально відрізняються. Якщо в традиційному середовищі розробки ви пишете цілісну програму, то Jupyter Notebook пропонує інтерактивний формат. Код виконується окремими блоками, і ви одразу бачите результат під кожним з них. Цей формат є абсолютним стандартом для аналізу даних, побудови графіків та створення наукових звітів, але він не підходить для створення класичних програм чи ботів.

Також варто згадати про Spyder. На перший погляд, це схожі середовища, проте Spyder створювався зі специфічним ухилом у наукову сферу. Він має спеціальні вбудовані вікна для перегляду великих масивів даних, побудови складних діаграм та виконання математичних обчислень. Тоді як PyCharm залишається більш універсальним інструментом для розробки загального програмного забезпечення.

Підсумовуючи це порівняння, можна сказати, що кожен з перелічених інструментів ідеально підходить для своїх специфічних завдань. Проте саме PyCharm найчастіше обирають ті фахівці, яким потрібен надійний та комплексний інструмент для створення, тестування та підтримки масштабних програмних проєктів на мові Python.

Мова програмування Python:

На даний момент це найпопулярніша скриптова мова. Мова швидкої розробки. Фактично, на ній можна швиденько написати будь-що. Нажаль не зможете ні на чому іншому розробляти так само швидко, як на Python. Мова проста для вивчення і проста у використанні. Тому у Python дуже багато сфер застосування. Найчастіше її

використовують там, де потрібно просто автоматизувати якісь дії, тому її люблять у різних професіях. Python використовують DevOps, Automation QA для автоматизованого тестування, Data Scientist та інші фахівці. Але потрібно розуміти, що Python використовується не там, де результатом роботи є код, вона застосовується як додатковий інструмент. Тобто, фахівець може зробити все руками або з використанням інших інструментів, які не зав'язані на коді. Наприклад, протестувати можна мануально, але зі скриптом виходить і надійніше, і швидше. Ось для цих автоматичних тестів може використовуватися Python.

Python також може використовуватися як програмістська мова, тобто, коли результатом спеціаліста є саме код. Однак у цьому випадку область застосування рівно одна – створення веб-застосунків. У цьому напрямку дуже популярний фреймворк Django. Є й інші фреймворки, але Python+Django – класичне поєднання. Python дозволяє створити сайти складніші, ніж WordPress. Як правило, це веб-сайти для e-commerce, будь-які сайти, які веде бізнес (записатися до лікаря, прослухати курси, взяти участь у якихось активностях тощо). Зробити таке на WordPress складно та дорого, тому для цього використовується Python, або його прямі конкуренти – PHP та Ruby.

На Python, як і будь-якою іншою сучасною мовою програмування, можна написати що завгодно. Але все ж таки є основні сфери застосування для тієї чи іншої мови. Наприклад, якщо розробляти програми під Windows, то це не про Python. Так, технічно можна їх розробити цією мовою, але це буде дуже рідкісне явище.

Щодо мови C#, то вона також має доволі широке поле для застосування. Проте, на відміну від Python, її майже ніколи не використовують як допоміжний скриптовий засіб. Загалом можна виділити три глобальні вектори, де цей інструмент розкривається найкраще.

Перший напрямок стосується створення великих корпоративних систем (так звана Enterprise-розробка), які допомагають автоматизувати бізнес-процеси величезних компаній. У цій сфері головним суперником виступає Java. Але C# отримує беззаперечну перевагу, якщо корпоративна інфраструктура замовника тісно пов'язана з документами Microsoft Office. Завдяки глибокій нативній інтеграції,

фінансовим, аналітичним та юридичним установам набагато вигідніше замовляти програмні продукти саме на цій технології.

Друга ніша – це конструювання десктопних програм безпосередньо під операційну систему Windows. Для цього зазвичай залучають такі платформи, як WinForms або WPF. Звісно, розробка класичних віконних застосунків сьогодні вже не є домінуючим напрямком, проте попит на фахівців у цій області досі зберігається.

Третій вектор сподобається тим, хто мріє про створення ігор. Мова йде про роботу з Unity – другим за популярністю ігровим рушієм у світі. Якщо розробника цікавить сфера GameDev, то цей шлях буде найбільш оптимальним.

Варто зазначити, що програмісти можуть відносно легко переходити між цими трьома спеціалізаціями, просто опановуючи нові відповідні фреймворки.

Якщо порівнювати складність навчання, то поріг входження в Python об'єктивно нижчий. Мова справді інтуїтивніша, тому величезна кількість новачків обирає саме її для старту своєї кар'єри. Як наслідок, на ринку багато початківців, і стартові зарплати молодших спеціалістів тут зазвичай трохи скромніші. Проте з набуттям досвіду та переходом на вищі посади рівень доходів програмістів у обох екосистемах практично зрівнюється.

Важлива відмінність полягає у підходах до типізації. Для Python характерна динамічна типізація. Це означає, що під час написання коду програмісту не потрібно жорстко вказувати, який саме тип даних зберігатиметься у змінній. Інтерпретатор самостійно визначає це безпосередньо в момент виконання команд. З одного боку, такий підхід значно пришвидшує процес розробки, але з іншого – може створити додаткові труднощі при побудові складної архітектури.

Натомість C# базується на суворій статичній типізації. Перед використанням кожної змінної розробник зобов'язаний чітко задекларувати її тип. Такий педантичний підхід робить процес написання коду дещо повільнішим, але натомість надає в руки програміста значно надійніший інструмент. Саме тому для реалізації масштабних та багатокомпонентних систем цей варіант підходить краще.

Цікаво, що візуально синтаксис обох технологій має певні спільні риси з класичною мовою C. Хоча офіційно Python до цієї родини не належить, адже замість

звичних фігурних дужок тут логічні блоки виділяються звичайними відступами. У будь-якому разі, щоб зробити остаточний вибір і зрозуміти власні вподобання, фахівцю бажано спробувати попрацювати з обома інструментами на практиці.

Щодо майбутніх перспектив, то Python продовжує стрімко нарощувати свою присутність на ринку. Незважаючи на свій поважний вік (мова була створена ще у 1991 році і є старшою за ту ж Java), вона здобула репутацію надзвичайно сучасного та молодіжного інструменту. Головний секрет успіху полягає в тому, що творці змогли ідеально задовольнити потреби цільової аудиторії. Спільноті був потрібен зручний, лаконічний засіб для максимально швидкого написання програм, і ця технологія стала ідеальною відповіддю на такий запит. Звісно, з часом загальний ажіотаж може трохи вщухнути, проте інструмент гарантовано збереже свою популярність. На сьогоднішній день просто не існує повноцінних конкурентів, які б могли настільки ж ефективно замінити його в сегменті швидкої розробки та автоматизації рутинних завдань.

Зважаючи на всі наведені вище аргументи, для реалізації практичної частини даної роботи була обрана саме мова Python, а робочим середовищем для написання коду виступила програма PyCharm.

1.4.2 Аналіз середовища розробки та допоміжних бібліотек

Наведено модулі які були використані в кваліфікаційній роботі, їх опис та приклад застосування:

Psycopg2:

– Опис: Використовується для взаємодії з базою даних PostgreSQL. Може бути корисним для зберігання даних про користувачів, чати, повідомлення тощо.

– Приклад: Створення системи управління користувачами, де зберігаються їхні профілі та історія повідомлень.

Telebot:

– Опис: Бібліотека для створення Telegram-ботів. Може бути використана для реалізації інтерактивних чат-ботів, які автоматично відповідають на запити користувачів.

– Приклад: Розробка бота для навчальних цілей, який надає інформацію про навчальні програми, розклад занять тощо.

Functools:

– Опис: Стандартні інструменти для роботи з функціями, зокрема для створення декораторів та оптимізації коду.

– Приклад: Використання для кешування результатів часто викликаних функцій для підвищення продуктивності системи.

Chat_repository:

– Опис: Власний модуль для управління даними чатів. Може бути корисним для організації та збереження інформації про чати та повідомлення.

– Приклад: Реалізація репозиторію для зберігання історії чатів між користувачами та ботом.

Mapper:

– Опис: Власний модуль для перетворення даних з одного формату в інший. Зазвичай використовується для відповідності між різними структурами даних.

– Приклад: Перетворення даних, отриманих від API, у формат, який зручно зберігати у базі даних.

Message_parser:

– Опис: Власний модуль для аналізу та обробки текстових повідомлень. Може бути використаний для виділення ключової інформації з повідомлень.

– Приклад: Аналіз повідомлень користувачів для визначення їхніх запитів та автоматичної відповіді на них.

Model:

– Опис: Власний модуль для визначення моделей даних, які використовуються в програмі. Це можуть бути користувачі, повідомлення, чати та інші сутності.

– Приклад: Визначення моделі користувача з атрибутами, такими як ім'я, вік, історія повідомлень тощо.

Prompt:

– Опис: Власний модуль для створення та обробки запитів до користувача. Використовується для взаємодії з користувачем через текстові запити.

– Приклад: Створення інтерфейсу для запитів до користувача, таких як введення його імені, вибір опцій тощо.

Startup:

– Опис: Власний модуль для ініціалізації та налаштування програми при запуску. Може виконувати початкові налаштування та конфігурації.

– Приклад: Ініціалізація бази даних, завантаження конфігураційних файлів та інших ресурсів при запуску програми.

Utility:

– Опис: Власний модуль з утилітами та допоміжними функціями. Використовується для виконання загальних операцій, необхідних в різних частинах програми.

– Приклад: Логування подій, обробка помилок, форматування даних та інші допоміжні функції.

Datetime:

– Опис: Стандартний модуль Python для роботи з датами та часом. Використовується для отримання поточної дати та часу, обчислення різниці між датами тощо.

– Приклад: Відстеження часу відправлення повідомлень, ведення журналу подій із зазначенням часу тощо.

Функція обробляє команду "/start" у чат-боті, і коли вона викликається, створюється новий запис про чат у базі даних з ідентифікатором чату:

Ця функція відповідає на команду "/current_day" у чат-боті. Вона оновлює стан чату, визначає поточний день тижня, отримує розклад для цього дня з бази даних та надсилає його користувачеві:

Ця функція відповідає на команду "/now" у чат-боті. Вона визначає поточний час та день тижня, а потім отримує поточний предмет з розкладу за допомогою chat_repository.get_current_subject. Якщо предмет не знайдено, бот надсилає повідомлення про відсутність занять на поточний день. Якщо поточний час знаходиться між

початком та закінченням предмету, бот надсилає повідомлення про те, який предмет зараз проходить. В іншому випадку бот надсилає повідомлення про наступний предмет:

Відповідає за обробку текстових повідомлень, які надіслав користувач до бота. Якщо стан чату визначений як "TIME_SLOT", бот спробує розпарсити повідомлення користувача для визначення предмету, який він хоче додати до розкладу.

Якщо вдалося розпарсити, бот перевіряє можливі конфлікти з іншими предметами у розкладі, якщо такі є. Якщо конфліктів немає, предмет додається до розкладу. Якщо стан чату визначений як "SUBJECT_POSITION", бот спробує розпарсити повідомлення користувача для визначення предмету, який потрібно видалити з розкладу за певним днем і позицією.

Якщо вдалося розпарсити, бот видаляє предмет з розкладу. У випадку будь-яких помилок або невдалих спроб розпарсити повідомлення, бот надсилає відповідне повідомлення користувачеві

Ця функція оновлює стан чату у базі даних. Вона приймає з'єднання з базою даних, ідентифікатор чату (chat_id) та новий стан (state). Функція використовує SQL-запит з конструкцією WITH для оновлення стану чату.

Якщо запис існує, він оновлюється; якщо ні, то він вставляється. Результатом є кількість змінених або вставлених записів, яка повертається в результаті запиту.

Ця функція призначена для розпізнавання тексту, який користувач надсилає з метою видалення предмету з розкладу. Вона використовує регулярний вираз subject_removal_regex, щоб визначити день тижня та позицію предмету у розкладі. Якщо текст відповідає формату очікуваного регулярного виразу, функція повертає об'єкт model.SubjectRemovalParseResult з визначеними днем тижня та позицією предмету, а також без помилок.

У протилежному випадку, якщо текст не відповідає очікуваному формату, функція повертає об'єкт model.SubjectRemovalParseResult з помилкою та без визначених дня тижня та позиції предмету.

1.4.3 Технології збереження даних та хмарні сервіси (Firebase/JSON)

Telegram– це не просто черговий месенджер, а ціла екосистема для спілкування та обміну файлами. Його створив програміст Павло Дуров, а за технічну частину та безпеку відповідав його брат Микола. Саме він розробив унікальний протокол шифрування MTProto, який захищає все: від ваших повідомлень у чатах до процесу входу в акаунт. Надійність системи підкріплюється тим, що сервери розкидані по всьому світу – від Сінгапура до Сан-Франциско, тому дані не зберігаються в одному місці.

Сервіс залишається доступним для всіх, оскільки його творці спочатку ставили за мету зробити вільний простір для зв'язку. Цікаво, що Telegram ніколи не витрачав кошти на масштабну рекламу чи телевізійні ролики – він став популярним завдяки сарафанному радіо та рекомендаціям самих користувачів. На сьогодні кількість людей у системі вже давно перевалила за сотні мільйонів, і щодня ця цифра зростає.

Ось що можна робити в Telegram:

- Спілкуватися з ким завгодно по всьому світу, використовуючи текстові чати або якісні голосові дзвінки.
- Пересилати важкі файли (фото, відео, архіви) розміром до 2 ГБ.
- Збирати величезні компанії у групах та супергрупах (до 200 000 людей), де адміністратори можуть гнучко налаштовувати правила та порядок.
- Знаходити людей за нікнеймом, навіть якщо ви не знаєте їхнього номера телефону.
- Підписуватися на канали: тут є все – від новин і політики до рецептів та дитячих казок.
- Слухати музику через вбудований плеєр, який працює навіть без інтернету, якщо пісні завантажені в кеш.
- Використовувати чати як безлімітне хмарне сховище для своїх фото чи заміток.
- Користуватися ботами – автоматичними помічниками, які можуть перекладати тексти, шукати квитки або розважати іграми.
- Головні плюси, за які його люблять:

– Швидкість. Повідомлення прилітають миттєво, навіть якщо інтернет не найкращий.

– Простота. Реєстрація займає хвилину – потрібен тільки телефон, а контакти підтягнуться самі.

– Універсальність. Програма працює всюди: на смартфонах, планшетах, комп'ютерах і навіть у терміналі Unix-систем.

– Повна свобода. Немає обмежень на кількість повідомлень чи обсяг завантажених у хмару даних.

Звісно, ідеальних програм не буває, і в Telegram теж є свої нюанси. Один із головних мінусів – це служба підтримки. Достукатися до них важко, а спілкуються вони зазвичай англійською або російською. Раніше була проблема і з українською мовою в самому додатку, доводилося ставити якісь сторонні файли для перекладу, але ще у 2018 році це виправили, тож зараз усе працює нормально і мову можна просто обрати в налаштуваннях.

Через те, що месенджер дуже сильно оберігає приватність і відмовляється видавати переписку силовикам, він став популярним не тільки серед звичайних людей, а й серед злочинців. Продаж наркотиків чи координація терористів – це та тіньова сторона, за яку компанію часто звинувачують у співучасті. Через неможливість контролювати цей потік інформації додаток час від часу блокують у різних країнах, якот у Китаї чи Ірані. Та й з анонімністю не все так просто: іноді можна дізнатися номер телефону користувача, що трохи псує картинку повної секретності профілю.

Щодо каналів, то це дуже зручна штука для блогів чи новин. Можна легко створити свій ресурс і наповнювати його будь-яким контентом. Користувач сам обирає, як їх читати: або отримувати повідомлення як від звичайних людей, або вимкнути звук і переглядати стрічку, коли з'явиться вільна хвилина. Канали бувають відкритими для всіх або приватними, куди пускають тільки за запитом адміністратора.

Окремої уваги заслуговують боти. Це такі собі віртуальні асистенти, які можуть виконувати купу завдань: шукати курс валют, перевіряти погоду чи конвертувати файли. Вони бувають двох видів. Звичайні боти працюють як окремий чат, де ви пишете команди. Інтегровані (інлайнові) боти дозволяють шукати інформацію

прямо під час переписки в іншому чаті – достатньо просто ввести назву бота, наприклад @gif, і ключове слово.

Зараз Telegram став потужним інструментом для бізнесу. Компанії використовують його для всього: консультують клієнтів, створюють вітрини з товарами і навіть приймають оплату через вбудовані системи на кшталт Stripe чи Paycom. Це дуже зручно, бо покупцю не треба виходити з додатка, щоб замовити річ. Оскільки аудиторія месенджера молода та просунута, заходити сюди зі своїм бізнесом зараз – дуже правильний крок. Тут є все: безпека, швидкість і величезний набір безкоштовних інструментів для роботи.

Telegram далеко не єдиний гравець на ринку, і у нього вистачає серйозних конкурентів. Якщо розібратися, то у кожного месенджера своя фішка. Наприклад, WhatsApp – це світовий лідер за кількістю людей, він зручний для дзвінків, хоча багатьох бентежить його тісний зв'язок із Facebook. Якщо ж вам потрібна максимальна анонімність, то зазвичай обирають Signal, де все зашифровано «до зубів». У нас в Україні досі дуже популярний Viber, особливо через його можливість дзвонити на звичайні номери. Є ще Facebook Messenger для тих, хто живе у соцмережах, або азійські гіганти на кшталт WeChat та LINE, які в своїх країнах замінюють людям взагалі все – від гаманця до ігрової консолі.

Сьогодні ми майже не випускаємо смартфони з рук. Це продиктовано часом і подіями, які ми зараз переживаємо: робота, навчання та спілкування з рідними повністю перейшли в онлайн. Telegram у цьому плані став чимось більшим, ніж просто чатиком. Окрім звичних стікерів, GIF-файлів та відеодзвінків, він пропонує круту штуку – ботів.

Взагалі боти в Telegram існують ще з 2015 року, але зараз вони реально спрощують побут. Мільйони людей через них конвертують файли, купують квитки чи навіть платять податки. Для бізнесу це взагалі знахідка: бот може приймати оплату через Apple Pay чи Google Pay, збирати відгуки та відповідати на питання клієнтів цілодобово. Це знімає купу роботи з живої підтримки.

Але справжній прорив стався навесні 2023 року. Те, що розробники назвали справжньою революцією, змінило правила гри. Завдяки використанню JavaScript

боти перестали бути просто набором кнопок у чаті. Тепер вони можуть виглядати як повноцінні вебсайти або додатки з гнучким інтерфейсом. По суті, тепер через звичайного бота можна відкрити цілу вітрину магазину з вибором товарів та анімаціями, яка нічим не поступається окремому сайту. Це робить Telegram справді універсальним інструментом для всього підряд.

Хоч ботів у Telegram зараз величезна кількість, майже всі вони мають спільний «скелет». Щоб «оживити» будь-якого з них, достатньо просто натиснути команду /start. Якщо ж бот наворочений і має багато функцій, їх список зазвичай ховають під кнопкою меню в лівому нижньому кутку екрана. Принцип роботи завжди один і той самий: ви кидаєте запит – бот видає результат.

З моменту своєї появи у 2015 році ці віртуальні помічники постійно розвивалися. Але справжній прорив, який називають революційним, стався 25 квітня 2023 року. Завдяки підтримці JavaScript боти тепер можуть мати такий вигляд і функціонал, що цілком здатні конкурувати з вебсайтами чи інтернет-магазинами. Це реально розширило межі того, що можна робити, не виходячи з месенджера.

Такі сервіси добряче полегшують життя, автоматизуючи все, що тільки можна – від онлайн-покупок до техпідтримки, що робить їх незамінними і для бізнесу, і для особистих справ. Ось лише кілька прикладів того, на що вони здатні:

- Рутина під контролем: боти нагадають про важливі події, допоможуть зі списком справ або швиденько перетворять файл з одного формату в інший.
- Швидкий пошук: замість шукати самому в інтернеті, можна просто спитати в бота – він і інформацію знайде, і на питання відповість.
- Шопінг і фінанси: через чат можна вибирати товари, здійснювати оплату, перевіряти баланс на картці або переказувати гроші.
- Бізнес-помічники: вони беруть на себе техпідтримку, оформлюють замовлення та розвантажують живих співробітників.
- Відпочинок: ігри, квізи та генерація мемів прямо в чаті – чудова штука, щоб розважитися.
- Колективні справи: організація групових подій, голосування за дату зустрічі чи розподіл обов’язків.

- Навчання та контроль: від навчальних матеріалів до моніторингу курсів валют, свіжих новин чи акцій.

- Персоналізація: бот підбирає контент на основі ваших інтересів та попередньої взаємодії.

Наведено більш структурований опис порівняльних переваг ботів у Telegram:

- Гнучкість і можливості програмування: Оновлення ботів у Telegram, що дозволяє використовувати мову програмування JavaScript, робить їх надзвичайно гнучкими та потужними. Це відкриває широкі можливості для розробників, які можуть створювати складні інтерфейси та функції, що конкурують з веб-сайтами.

- Спрощений користувацький інтерфейс: Один з головних плюсів ботів у Telegram - це їхній стандартизований користувацький інтерфейс. Це спрощує використання ботів для користувачів, оскільки вони можуть швидко зрозуміти, як взаємодіяти з будь-яким ботом.

- Інтеграція з платіжними системами: Можливість інтеграції з різними платіжними провайдерами, такими як Google Pay та Apple Pay, робить ботів у Telegram зручними для здійснення платежів прямо через месенджер.

- Різноманітність функцій: Боти у Telegram можуть бути використані для широкого спектру завдань, від автоматизації до комунікації з бізнесом. Це робить їх універсальним інструментом для користувачів з різними потребами.

- Зручність використання: Оскільки боти у Telegram інтегровані безпосередньо в популярний месенджер, вони є дуже зручними для використання. Користувачам не потрібно встановлювати додаткові програми або переходити на інші платформи, що робить їх особливо привабливими.

1.5 Об’єкт, предмет, мета та постановка завдань кваліфікаційної роботи

Будь-яка інженерна чи наукова розробка починається з того, що потрібно розставити всі крапки над «і» в плані методології. Щоб програма справді працювала, а результати можна було адекватно оцінити, важливо чітко бачити межі дослідження, розуміти головну ціль і розбити її на конкретні практичні кроки. Оскільки ця робота

зосереджена на тому, щоб спростити доступ до інформації в університетах, методологічна база виглядає наступним чином.

Об’єктом дослідження я обрав те, як саме в сучасних вишах організоване інформування та управління навчальним процесом, особливо з огляду на загальну цифровізацію. Сюди входить усе: як адміністрація, викладачі та студенти взаємодіють між собою, як вони дізнаються про розклад і як швидко ці дані оновлюються. Фактично, я аналізую перехід від старих статичних методів до сучасних автоматизованих систем, де обмін даними відбувається динамічно.

Предметом дослідження стали конкретні технічні інструменти: алгоритми, архітектурні патерни та методи розробки систем на базі технології чат-ботів, зокрема на платформі Telegram. Основний акцент тут – на створенні швидкого, безпечного та персоналізованого доступу до баз даних із розкладом занять.

Головна мета цієї роботи – зробити так, щоб студенти та викладачі могли отримувати академічну інформацію значно швидше та зручніше. Для цього я планую спроєктувати та запустити в роботу спеціальний Telegram-бот. Це дозволить студентам витрачати менше часу на пошуки актуальних пар, прибере зайві бар'єри в комунікації при змінах у навчанні, а також розвантажить офіційні сайти закладу, які часто перевантажені в пікові години.

Для досягнення поставленої мети та успішної реалізації програмного продукту було визначено та сформульовано наступний комплекс взаємопов'язаних завдань:

1. Проведення системного аналізу предметної області: дослідити проблематику традиційних підходів до поширення розкладу, проаналізувати існуючі програмні рішення (LMS-системи, мобільні застосунки, цифрові календарі), виявити їхні архітектурні та ергономічні недоліки задля формування обґрунтованої концепції власного продукту.

2. Формування вимог до програмного забезпечення: визначити функціональні (отримання розкладу на день/тиждень, вибір параметрів групи) та нефункціональні (висока продуктивність, стійкість до навантажень, кросплатформеність) вимоги до системи. Обґрунтувати вибір технологічного стеку, зокрема мови програмування (Python), механізмів асинхронної обробки подій (asyncio) та бібліотек для взаємодії

з Telegram Bot API.

3. Проєктування архітектури та моделі даних: розробити структуру реляційної бази даних (на базі SQLite/PostgreSQL) для ефективного зберігання інформації про факультети, курси, групи, дисципліни, викладачів та часові слоти. Спроєктувати логіку взаємодії користувача з системою із застосуванням архітектурного патерну скінченного автомата (Finite State Machine) для надійного управління станами діалогу.

4. Програмна реалізація серверної частини (бекенду): написати вихідний код програми, реалізувати об'єктно-орієнтовані моделі сутностей предметної області (дні тижня, події, користувачі, стани чатів), налаштувати маршрутизацію (роутинг) вхідних повідомлень та запитів від інтерактивних кнопок (callback queries).

5. Розробка користувацького інтерфейсу (Conversational UI): спроєктувати інтуїтивно зрозуміле дерево навігації в месенджері за допомогою системи дворівневих клавіатур (Reply та Inline), що мінімізує необхідність ручного введення тексту користувачем та унеможливлює синтаксичні помилки під час запитів.

6. Тестування та розгортання системи: провести функціональне тестування розробленого модуля на предмет коректного парсингу та видачі розкладу, перевірити стійкість логіки FSM до нестандартних дій користувача. Підготувати програмний продукт до розгортання на серверному обладнанні та інтеграції в реальне освітнє середовище.

Вирішення вищезазначених завдань у повному обсязі дозволить створити завершений, конкурентоспроможний та практично значущий програмний продукт, який повністю відповідає сучасним стандартам розробки інтерфейсів взаємодії та потребам цифровізації закладів вищої освіти.

2 . ПРОЕКТУВАННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ РОЗКЛАДУ

2.1 Формування функціональних та нефункціональних вимог

Процес розробки будь-якого програмного забезпечення починається з детального аналізу предметної області та специфікації вимог. Для створення надійного Telegram-бота, призначеного для надання доступу до навчального розкладу, необхідно чітко розмежувати функціональні можливості (що саме повинна робити система) та нефункціональні характеристики (як саме система має працювати, якими атрибутами якості володіти). Формалізація цих параметрів дозволяє уникнути архітектурних помилок на ранніх етапах та забезпечити відповідність готового продукту очікуванням кінцевих користувачів.

Функціональні вимоги

Функціональні вимоги описують специфічну поведінку системи, її реакції на вхідні дані та перелік операцій, які вона здатна виконувати. Виходячи з логіки роботи розкладу та взаємодії через месенджер, до програмного ядра бота висуваються наступні вимоги:

1. Реєстрація та ідентифікація користувача. Система повинна обробляти вхідні події (в базовій архітектурі представлені класом Event), витягуючи з них метадані про користувача: ідентифікатор повідомлення, унікальний ідентифікатор відправника (from_id), його ім'я, прізвище, username та ідентифікатор чату (chat_id). Це необхідно для створення персоналізованої сесії та прив'язки конкретного розкладу до профілю студента в базі даних.

2. Управління станами діалогу (Скінченний автомат). Оскільки взаємодія з ботом не є лінійною і передбачає багатокрокову навігацію, система має підтримувати механізм збереження поточного стану чату (ChatState). На програмному рівні це вимагає реалізації перерахувань (Enum), наприклад, базових станів: START (початковий екран), TIME_SLOT (вибір часового проміжку або пари), SUBJECT_POSITION (вибір конкретної дисципліни або її позиції в розкладі). Бот

повинен перевіряти стан користувача перед обробкою будь-якого текстового повідомлення, щоб уникнути некоректних спрацювань команд.

3. Обробка та виведення навчальних дисциплін. Головною функцією є надання структурованої інформації про розклад. Програма має оперувати сутністю навчальної дисципліни (Subject), яка обов'язково повинна містити такі атрибути: день тижня (day), назва предмету (title), час початку (start) та час закінчення (finish). Користувач повинен мати можливість запросити розклад як на конкретний день, так і на весь тиждень.

4. Парсинг та валідація даних. Оскільки дані про розклад можуть надходити у вигляді тексту або імпортуватися із зовнішніх джерел, система повинна реалізовувати механізм їх обробки. Вимагається наявність модуля валідації (наприклад, сутність SubjectParseResult), який перевірятиме коректність вхідних даних про предмет та у разі виявлення помилок (неправильний формат часу, відсутність назви) генеруватиме відповідне повідомлення про помилку замість аварійного завершення роботи програми.

5. Навігація за днями тижня. Інтерфейс користувача має містити меню для швидкого доступу до днів. На програмному рівні система повинна підтримувати чітку класифікацію днів за допомогою перерахувань (від MONDAY до SUNDAY), а також передбачати резервний варіант (UNRECOGNIZED) для обробки нестандартних запитів або збоїв при пошуку в базі розкладу.

Нефункціональні вимоги:

Нефункціональні вимоги стосуються архітектурних особливостей продукту, його надійності, продуктивності та безпеки. Для Telegram-бота розкладу визначено такі ключові параметри:

1. Продуктивність та час відгуку. Бот належить до класу застосунків з високою інтенсивністю операцій вводу-виводу (I/O-bound). Особливістю навчального розкладу є яскраво виражені пікові навантаження – максимальна кількість запитів надходить у ранкові години перед початком пар (07:30 - 08:30). Час реакції системи на базову команду (відправка повідомлення з розкладом) не повинен перевищувати 1-2 секунд навіть за умови одночасного звернення кількох сотень користувачів. Для

цього архітектура повинна підтримувати асинхронну обробку подій.

2. Масштабованість. Структура коду повинна бути побудована за принципами об'єктно-орієнтованого програмування з використанням незалежних модулів. Сутності "Користувач", "Подія", "Стан чату" та "Дисципліна" мають бути логічно розділені. Це дозволить у майбутньому легко розширити функціонал – наприклад, додати можливість прикріплення домашніх завдань до конкретного об'єкта Subject без переписування всього ядра бота.

3. Ергономічність (Usability). Взаємодія з ботом має бути максимально спрощеною і не вимагати від користувача запам'ятовування складних текстових команд. Основним інструментом взаємодії повинні стати Inline-кнопки під повідомленнями та Reply-клавіатури. Ручне введення тексту має бути зведено до абсолютного мінімуму (переважно для етапу пошуку або введення специфічних даних), що повністю нівелює проблему друкарських помилок з боку користувача та спрощує логіку обробки вхідного тексту.

4. Надійність і стійкість до відмов. Робота програми критично залежить від доступності серверів Telegram (Bot API). Відповідно, мережевий шар програми повинен містити обробники таймаутів (timeout exceptions) та механізми автоматичного повторення запитів (retries) у разі тимчасової втрати з'єднання з API месенджера. Робота програми не повинна аварійно завершуватися, якщо сервер Telegram повертає помилки 502 або 504.

5. Безпека даних. Хоча навчальний розклад є публічною інформацією, система повинна захищати базу даних від ін'єкцій та несанкціонованого доступу. Усі вхідні текстові дані (text у класі Event) повинні проходити санітизацію перед їх використанням у SQL-запитах або логіці парсингу. Крім того, внутрішня структура файлів проєкту має передбачати винесення конфіденційних даних (API-токен бота, конфігурації підключення до БД) у змінні середовища (Environment variables), щоб унеможливити їх випадкову публікацію в системах контролю версій.

2.2 Побудова діаграм сценаріїв використання (Use Case)

Створення діаграм UML допомагає краще зрозуміти та проаналізувати логіку програмного забезпечення, як працює програмне забезпечення, показуючи всі залежні компоненти та об’єкти, їх взаємодію та функціональність програмного забезпечення.

Представляют статично структуру додатку 4 типи діаграм;

– Представляють поведінкові аспекти системи 5 типів діаграм;

– Представляють фізичні аспекти функціонування системи 3 типи діаграм.

Деякі види діаграм є специфічними для певних додатків. Найдоступнішими з них:

– Діаграма прецедентів (Use-case diagram);

– Діаграма класів (Class diagram);

– Діаграма активностей (Activity diagram);

– Діаграма послідовності (Sequence diagram);

– Діаграма станів (Statechart diagram);

Діаграма прецедентів – Use-case diagram:

Діаграма прецедентів або діаграма прецедентів використання в UML – це діаграма, яка показує зв’язки між акторами та прецедентами, і є компонентом моделі прецедентів, яка дозволяє описати систему на концептуальному рівні.

Actor – це зовнішній фактор (поза системою), який взаємодіє з системою, беручи участь (і зазвичай ініціюючи) варіанти використання. На практиці акторами можуть бути звичайні люди (наприклад, користувачі системи), інші комп’ютерні системи або зовнішні події.

Use Case – це послідовність операцій, сценарій, за яким користувач взаємодіє з додатком, програмою, яка виконує будь-які дії для досягнення певної мети. Тестування варіантів використання проводиться для виявлення інших логічних лазівок і помилок у програмі, які важко знайти під час тестування кожного модуля, кожної частини програми окремо.

Class Diagram - Діаграма класів:

Діаграма класів – це структурна діаграма мови моделювання UML, яка показує загальну структуру ієрархії класів системи, їх взаємодію, властивості, методи, інтерфейси та їхні зв’язки.

Клас – це набір речей, які мають спільні властивості та операції. Сама діаграма класів є статичним набором елементів декларативної моделі. Він надає найбільш повне та детальне уявлення про зв’язки між програмним кодом, функціональністю та інформацією про окремі класи.

Діаграма діяльності:

Діаграма діяльності – в UML, візуальне представлення діаграми діяльності. Діаграма діяльності – це діаграма стану кінцевого автомата, вершинами якої є конкретні дії та переходи, які відбуваються після завершення дії.

Діаграми діяльності описують динамічні аспекти поведінки системи у формі блок-схеми, що відображає бізнес-процеси, логіку процесу та робочий процес (переходи від однієї діяльності до іншої).

Діаграма послідовності - Sequence diagram:

Діаграма послідовності – показує взаємодії об’єктів, упорядкованих у часі. Зокрема, такі діаграми показують залучені об’єкти та послідовність надісланих повідомлень.

Іншими словами, діаграми послідовності відображають часові характеристики об'єктів, що надсилають і отримують повідомлення.

Діаграма послідовності – Sequence diagram:

Діаграма послідовності – показує часові характеристики об'єкта, що передає та приймає повідомлення. Хронологію слід розуміти як послідовність операцій, і її не слід плутати з шкалою часу.

Діаграма стану – Statechart diagram:

Діаграма станів показує скінченний автомат у вигляді графа, вершинами якого є стани об’єкта, поведінка якого моделюється, а переходами – події, що переводять розглядуваний об’єкт з одного стану в інший.

Діаграми станів – це, по суті, діаграми станів із теорії автоматів із стандартизованою нотацією, яка може визначати різноманітні системи – від

комп’ютерних програм до бізнес-процесів. Використовуються такі умовні позначення: Кружки, що представляють початковий стан.

Діаграми UML (Unified Modeling Language) відіграють ключову роль у процесі розробки програмного забезпечення, допомагаючи командам програмістів і розробникам краще розуміти, планувати та реалізовувати програмне забезпечення. Як продовження різних типів діаграм UML, ось деяка додаткова інформація:

Діаграма компонентів (Component Diagram):

На схемі зображено компоненти системи та залежності між ними. Компонентами можуть бути програмні модулі, бібліотеки, файли, апаратні установки тощо. Це допомагає визначити, які частини системи можна повторно використовувати та як вони взаємодіють.

Діаграма Розгортання (Deployment Diagram):

Діаграма показує фізичну структуру системи, вказуючи, які компоненти розгортаються на яких апаратних пристроях або середовищах виконання. Це корисно для планування розгортання програмного забезпечення на серверах, комп’ютерах або хмарних службах.

Діаграма Об'єктів (Object Diagram):

Діаграми об’єктів показують конкретні об’єкти в системі та зв’язки між ними в певний момент часу. Це корисно для тестування конкретних сценаріїв або виявлення проблем із впровадженням.

Загалом, використання різних типів діаграм UML може допомогти командам розробників краще зрозуміти потреби клієнтів, розробити оптимальні рішення та спростити складність реалізації програмного забезпечення. Це забезпечує ефективну комунікацію між учасниками роботи та сприяє успішному виконанню проектних завдань.

Діаграми прецедентів будуть використані під час кваліфікаційних зусиль. Граф прецедентів – це граф, що складається з набору акторів, прецедентів, обмежених системними кордонами, і асоціацій між акторами та прецедентами. Діаграма варіантів представляє елементи моделі варіантів використання.

Суть діаграми випадків полягає в тому, що система, що проектується,

представлена у вигляді набору сутностей або акторів, які взаємодіють із системою через так звані варіанти використання. Варіанти використання використовуються для опису послуг, які система надає учасникам.

1. Визначте учасників.

2. Визначити варіанти використання.

3. Ранжуйте кожен випадок використання.

На рисунку 2.1 зображено діаграму прецедентів для телеграм-бота “Schedule” , що розробляється у рамках кваліфікаційної роботи.

Рис. 2.1 Діаграма варіантів використання

2.3 Проектування архітектури бота та станів діалогу (FSM)

Алгоритм – набір інструкцій, що описує послідовність дій виконавця для досягнення результату розв'язання задачі за допомогою скінченної кількості дій. Для візуалізації алгоритмів часто використовують блок-схеми.

Термін "алгоритм" походить від імені іранського математика і астронома Хорезмі, який написав трактат, що описує десяткову систему числення, винайдену в Індії близько 825 року.

Алгоритми також можна знайти в інших предметах. У хімії, наприклад, утворення сполук можна описати за допомогою алгоритмів. Однак найбільше прикладів алгоритмів можна знайти в математиці - науці, звідки власне і зародилося

це поняття. Фактично, математика аналізує різні алгоритми та генерує нові. Алгоритми, які вивчають у шкільному курсі математики, включають арифметичні дії та правила розв'язування рівнянь. Алгоритми можна використовувати для формулювання правил побудови різних геометричних фігур (згадаймо архітектурні задачі) або пропозицій щодо вирішення типових проблем.

Термін "алгоритм" схожий на терміни "метод" або "рецепт". Однак алгоритми в комп'ютерних науках - це не просто рецепти розв'язання проблем. Алгоритми в першу чергу розробляються для автоматизації дій виконавця.

Створення алгоритму починається з декомпозиції визначеного процесу на ряд окремих кроків. Властивість декомпозиції алгоритму на окремі кроки називається дискретизацією алгоритму. Кожен крок алгоритму формулюється у вигляді інструкції (команди), тобто конкретної вказівки виконавцю. Наприклад, вказується послідовність дій, які потрібно виконати для обчислення виразу A x X = B.

Особливості виконавців та алгоритмів.

Алгоритми розв'язання однієї і тієї ж задачі можуть бути представлені по-різному. Якщо ви хочете навчити собаку чогось робити, дайте словесну команду зрозумілою для неї мовою. Якщо ви вчите друга їздити на велосипеді, система команд, які він може виконувати, звичайно, ширша. Алгоритм їзди можна пояснити усно або на папері.

Алгоритми створюють, орієнтуючись на конкретного виконавця алгоритму, наприклад, на навчену тварину, людину, автоматизований пристрій або комп'ютер. Алгоритм повинен містити команди, які можуть бути зрозумілі та реалізовані виконавцем.

Характеристики алгоритмів.

1. Скінченність - виконання кожного алгоритму має бути завершено за скінченну кількість кроків.

2. Достовірність - виконання алгоритму завжди повинно призводити до фіксованого результату. Воно не повинно припинятися або ніколи не припинятися за невизначених умов.

3. Формальність - виконавець повинен слідувати алгоритму і отримувати

результат, не вникаючи в його суть. Ця властивість особливо важлива при автоматичному виконанні алгоритмів.

4. Ясність - будь-який алгоритм повинен бути визначений таким чином, щоб у виконавця не виникало неоднозначних інструкцій під час виконання. Іншими словами, різні виконавці повинні діяти однаково за алгоритмом і досягати однакових результатів.

5. Масовість - алгоритм може бути використаний для вирішення цілого класу задач

Дія алгоритму представляється у вигляді послідовності двох (або більше) простих операцій, що виконуються по черзі. Якщо алгоритм складається лише з послідовності простих операцій, він називається простим або лінійним алгоритмом.

Операція розгалуження - це інструкція, яка виконує одну з двох інструкцій, інструкцію 1 або інструкцію 2, залежно від того, чи є вираз P істинним або хибним. Якщо вираз P є хибним, виконується інструкція2. Окремим випадком розгалуження є неповне розгалуження, коли жодна операція не виконується, якщо вираз P є хибним.

Рис. 2.2 Схема роботи модуля

Структура повторення вказує на те, що послідовність команд повторюється певну кількість разів або до тих пір, поки не будуть виконані певні умови.

На рисунку 2.2 зображено схему алгоритму роботи модуля , що розробляється в рамках кваліфікаційної роботи.

2.4 Розробка структури бази даних для збереження прогресу

Діаграма класів – це статичне представлення структури моделі в UML. Представляє статичні елементи, такі як класи, типи даних, їхній вміст і зв'язки. Діаграма класів може містити представлення пакета або представлення вкладених пакетів.

Діаграма класів – це структурна діаграма, яка використовується для моделювання об'єктно-орієнтованих систем. Вони показують класи, їхні властивості, методи та зв'язки між класами. Діаграми класів дозволяють розбити систему на окремі компоненти та зрозуміти, як вони взаємодіють один з одним.

На рисунку 2.4 зображено схему діаграма класів, що розробляється в рамках кваліфікаційної роботи.

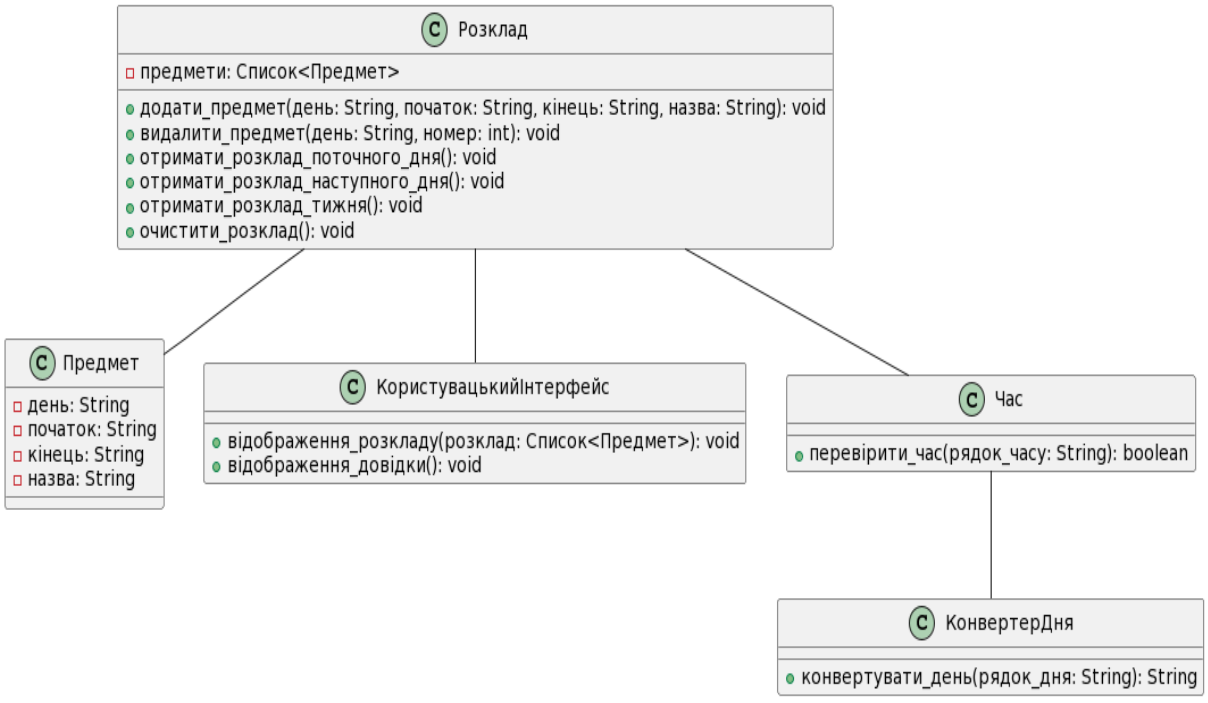

Рис. 2.4 Діаграма класів

2.5 Моделювання логіки парсингу та відображення занять

Основою функціонування будь-якої інформаційної системи, що працює з розкладом, є механізми обробки, перетворення та подальшого візуального представлення даних. У контексті розробки Telegram-бота для доступу до навчального розкладу, архітектура програмного продукту вимагає чіткого розділення логіки на два ключові етапи: парсинг (синтаксичний аналіз та структурування вхідних даних) та генерацію вихідного представлення (відображення розкладу кінцевому користувачеві). Для забезпечення високої надійності та стійкості системи до помилок, моделювання цих процесів базується на об'єктно-орієнтованій парадигмі з використанням суворо типізованих структур даних.

Першим етапом є побудова моделі даних предметної області. Атомарною одиницею навчального процесу є окреме заняття (пара). У програмному коді ця сутність інкапсулюється у вигляді класу Subject. Даний клас виступає контейнером, що зберігає вичерпну інформацію про конкретну дисципліну. До його основних атрибутів належать: day (день тижня, до якого прив'язане заняття), title (повна назва дисципліни, яка може включати тип заняття - лекція, практика чи лабораторна робота, а також ПІБ викладача та номер аудиторії), start (час початку заняття) та finish (час його закінчення). Створення окремого об'єкта для кожного заняття дозволяє системі гнучко маніпулювати масивами даних: сортувати їх за часом, фільтрувати за днями тижня або здійснювати пошук за назвою предмета.

Для уникнення колізій при обробці текстових даних, зокрема пов'язаних із різним написанням днів тижня (використання різних регістрів, скорочень або помилок вводу), логіка парсингу спирається на перерахування (Enumeration). Клас Day (наслідуваний від базового класу Enum) жорстко регламентує набір допустимих значень для днів тижня від MONDAY до SUNDAY. Критично важливим архітектурним рішенням є впровадження додаткового стану UNRECOGNIZED. Цей стан відіграє роль запобіжника під час синтаксичного аналізу: якщо алгоритм не може однозначно ідентифікувати день тижня у вхідному масиві даних, запису присвоюється статус

UNRECOGNIZED, що сигналізує системі про необхідність ручного коригування даних адміністратором або виведення відповідного повідомлення про помилку, замість непередбачуваної поведінки чи аварійного завершення роботи скрипта.

Процес парсингу вхідної інформації є потенційно вразливим місцем системи, оскільки джерело даних (текстовий файл, API університету або ручне введення) може містити неформатований текст, пропущені поля або некоректні формати часу. Тому моделювання логіки парсингу передбачає використання патерну "Result Wrapper" (обгортка результату). Для цього реалізовано клас SubjectParseResult, який приймає два параметри: subject (екземпляр розпарсеного об'єкта, якщо операція пройшла успішно) та error (опис помилки або флаг її наявності). Під час обробки кожного рядка розкладу, парсер застосовує регулярні вирази (RegEx) або методи розділення рядків (string splitting) для вилучення часу та назви предмета. Якщо рядок відповідає очікуваному шаблону, створюється об'єкт Subject, який обгортається в SubjectParseResult з порожнім полем помилки. У разі порушення формату об'єкт Subject залишається порожнім, а поле error заповнюється відповідним кодом виключення. Це гарантує безперебійну роботу бота навіть при частково пошкоджених вхідних даних бази розкладу.

Взаємодія системи з користувачем та ініціація процесів парсингу або запиту даних описується через модель подій. Клас Event акумулює всі необхідні метадані про поточний запит: унікальний ідентифікатор повідомлення (message_id), дані відправника (from_id, from_first_name, from_last_name, from_username), ідентифікатор діалогу (chat_id) та безпосередньо текстове тіло запиту (text). Цей об'єкт передається до маршрутизатора (роутера) бота, який, зважаючи на поточний стан чату (визначений у класі ChatState), приймає рішення щодо подальшої обробки події.

Логіка відображення занять є фінальним етапом обробки запиту користувача. Вона відповідає за зворотне перетворення масиву об'єктів Subject у читабельний та естетично оформлений текст, який підтримується інтерфейсом Telegram. Алгоритм відображення працює за наступним сценарієм:

1. Запит до бази даних або кешу формує вибірку об'єктів Subject для конкретного об'єкта Day (або списку днів days, якщо запитується розклад на тиждень).

2. Система перевіряє масив на наявність елементів. Якщо масив порожній, алгоритм генерує стандартизоване повідомлення про відсутність занять (наприклад, "На цей день занять не заплановано"), що є важливою вимогою для якісного користувацького досвіду (UX).

3. Якщо масив містить дані, алгоритм ініціює цикл, проходячи по кожному об'єкту Subject. Дані форматуються з використанням синтаксису HTML або MarkdownV2, що підтримується Telegram Bot API. Часові слоти (start - finish) виділяються жирним шрифтом для візуального акцентування, а назви предметів (title) розміщуються з нового рядка з використанням відповідних маркерів або емодзі для кращого візуального структурування тексту.

4. Сформовані текстові блоки конкатенуються в єдине повідомлення та відправляються користувачеві у чат за допомогою методу відправки повідомлень Telegram API.

Таким чином, запропонована модель логіки парсингу та відображення забезпечує чітку декомпозицію процесів. Використання класів Event та ChatState дозволяє ефективно керувати потоком виконання, класи Subject та Day гарантують консистентність даних усередині системи, а імплементація SubjectParseResult забезпечує високий рівень відмовостійкості програмного продукту при роботі з непередбачуваними вхідними даними.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ БОТА

3.1 Загальний опис структури програмного коду

Програмна частина нашого бота, який отримав назву Schedule, була повністю написана на мові Python. У процесі розробки я спирався на об’єктно-орієнтований підхід, що дозволило зробити код модульним. Це означає, що всі структури даних, внутрішня логіка та механізми спілкування з Telegram API чітко розділені між собою і не змішуються в одну купу.

В основі системи лежить кілька важливих компонентів. По-перше, для стабільної роботи ми використали стандартну бібліотеку enum. Це допомогло створити перерахування для днів тижня та різних станів діалогу. Наприклад, спеціальний клас Day відповідає за правильне розпізнавання днів, а клас State керує логікою скінченних автоматів. Саме завдяки цьому бот розуміє, на якому етапі розмови він зараз перебуває – чи це самий початок, чи момент вибору конкретного часу в розкладі.

Окрему увагу я приділив моделям сутностей. Для цього в коді прописані спеціальні класи, що описують реальні об'єкти. Клас Subject зберігає всю інформацію про навчальну дисципліну, включаючи час початку та кінця пари. Є також клас Event, який збирає всі дані про вхідне повідомлення від користувача, та ChatState, що дозволяє боту «запам’ятовувати», про що йшла мова в конкретному чаті. Це дуже важливо для підтримки безперервної сесії.

Щоб система працювала надійно і не ламалася при зчитуванні даних, я впровадив спеціальні класи для парсингу, зокрема SubjectParseResult. Вони перевіряють, чи все гаразд із даними у файлах формату JSON, звідки бот бере розклад. Це робить програму стійкою до дрібних помилок у базі даних чи випадкових некоректних запитів.

Весь процес обробки повідомлень відбувається в асинхронному режимі. Спеціальні обробники, або хендлери, постійно чекають на нові сигнали від сервера. Як тільки приходить повідомлення, бот миттєво визначає поточний стан чату, запускає потрібну функцію і повертає студенту відповідь – або звичайний текст, або набір кнопок. Така архітектура дуже зручна, бо вона дозволяє легко додавати нові функції, наприклад систему сповіщень чи різні рівні доступу для користувачів, без переписування всього ядра програми. Це робить кінцевий продукт гнучким і відкритим для подальшого вдосконалення.

3.2 Алгоритмічна реалізація ключових функцій

Фундаментом розробленого програмного забезпечення є сукупність алгоритмів, що забезпечують обробку запитів користувача в реальному часі та взаємодію з базою даних розкладу. Основна складність реалізації полягала у створенні відмовостійкої системи навігації та механізму обробки часових інтервалів.

3.2.1 Обробка команд користувача та навігація по днях тижня

В основі взаємодії лежить алгоритм диспетчеризації повідомлень. Процес обробки кожного запиту можна розділити на наступні етапи:

1. Ідентифікація події: Кожне вхідне повідомлення інкапсулюється в об’єкт класу Event. Алгоритм зчитує chat_id для визначення поточного стану користувача через екземпляр класу ChatState.

2. Мапінг станів (FSM): Використовуючи перерахування State, система визначає, на якому етапі діалогу знаходиться студент. Якщо стан дорівнює START, алгоритм активує головне меню.

3. Логіка вибору дня: При отриманні назви дня тижня, спрацьовує алгоритм валідації через перерахування Day. Якщо введений текст збігається з елементом списку days, система ініціює запит до сховища даних.

4. Фільтрація та вивід: Алгоритм здійснює ітерацію по всіх зареєстрованих

об'єктах Subject. Предмети, поле day яких відповідає запиту, додаються до динамічного списку, сортуються за часом початку (start) і трансформуються у фінальне текстове повідомлення.

3.2.2 Реалізація механізму сповіщень про заняття

Щоб бот міг самостійно стежити за часом і вчасно попереджати про початок занять, я реалізував спеціальний алгоритм фонового моніторингу. Працює це досить просто: у системі запускається асинхронна задача, яка прокидається щохвилини і робить швидку перевірку. Програма бере точний час сервера і порівнює його з графіком на сьогодні. Для кожної пари обчислюється часова різниця, яку в коді ми називаємо delta. Фактично це результат віднімання поточного моменту current_time від часу початку лекції subject.start. Коли цей зазор стає рівним встановленому ліміту, наприклад 15 хвилин, система автоматично запускає процес сповіщення.

Сама процедура розсилки теж відбувається в асинхронному режимі. Бот бере список усіх користувачів, які підписані на оновлення, і відправляє їм повідомлення про наступну пару. Тут є важливий технічний момент: щоб захистити сервіс від блокування з боку API Telegram через занадто велику кількість запитів одночасно, я додав механізм обмеження швидкості, або rate limiting. Програма робить невеликі паузи між пакетами даних, тому розсилка проходить стабільно і не перевантажує канали зв'язку месенджера.

3.3 Обробка виняткових ситуацій та валідація даних

Ще одним критично важливим моментом у роботі бота є те, як він поводиться з даними. Щоб програма не «падала» через будь-яку помилку в тексті розкладу, я розробив систему перевірки на основі класу SubjectParseResult. Це свого роду запобіжник, який спрацьовує кожного разу, коли бот намагається прочитати інформацію про нову пару.

Сам процес валідації влаштований досить ретельно. Коли система зчитує дані, вона відразу перевіряє, чи правильно записаний час і чи всі необхідні поля на місці.

Якщо раптом у базі щось заповнено не за форматом або бракує важливої інформації, алгоритм діє дуже обережно. Він не припиняє роботу всього бота, а створює спеціальний об'єкт, у якому виставляється прапорець помилки.

Цей об'єкт потім потрапляє до головного циклу програми. Бот «бачить», що сталася проблема, і замість того, щоб видати порожнє повідомлення або просто замовкнути, він надсилає користувачеві чітке сервісне повідомлення. У ньому вказується, що в базі даних знайдено помилку. Такий підхід робить сервіс набагато надійнішим, адже навіть якщо адміністратор випадково помилиться в одному рядку розкладу, всі інші функції бота працюватимуть стабільно.

3.4 Розробка користувацького інтерфейсу

Для розробки цього програмного комплексу було обрано кілька перевірених інструментів, які забезпечують стабільну роботу та високу швидкість обробки даних. Основою взаємодії з базою даних PostgreSQL став драйвер Psycopg2, який надійно керує всіма транзакціями, забезпечує збереження інформації про користувачів, їхні профілі та історію взаємодії без жодних збоїв чи втрат. Сам інтерфейс чат-бота побудований на базі популярної бібліотеки Telebot, відомої також як pyTelegramBotAPI, що дозволяє легко налаштувати маршрутизацію повідомлень, створювати зручні інтерактивні меню з кнопками та миттєво відповідати на запити студентів щодо розкладу чи навчальних програм. Окрім зовнішніх бібліотек, вагому роль відіграють і стандартні засоби мови Python, зокрема модуль Functools, який використовується для оптимізації коду та створення декораторів, що допомагає кешувати результати частих запитів і суттєво знижує навантаження на сервер. Нарешті, для точної синхронізації всіх процесів із реальним часом застосовується модуль Datetime, завдяки якому система безпомилково визначає поточну або наступну пару, відстежує точний час відправлення повідомлень та веде чіткий журнал подій. Всі ці компоненти працюють як єдине ціле, гарантуючи швидкість відгуку системи та її високу надійність під час щоденного використання.

3.5 Реалізація асинхронної обробки запитів (aiogram/telebot)

Будь-який Telegram-бот за своєю природою – це класичний I/O-bound застосунок. Це означає, що програма витрачає левову частку часу не на складні математичні обчислення (навантаження на процесор), а на банальне очікування: очікування відповіді від серверів Telegram, очікування результатів SQL-запиту від бази даних PostgreSQL або очікування запису логів на диск.

Якщо реалізовувати такого бота за класичною синхронною моделлю, ми стикаємося з серйозною архітектурною проблемою – блокуванням потоку виконання (blocking operations). У базовій реалізації бібліотеки pyTelegramBotAPI (відомої як telebot), коли бот отримує повідомлення і робить запит до бази даних, весь робочий потік зупиняється. Якщо в цей момент інший користувач надішле команду, системі доведеться або ставити його в чергу, або породжувати новий системний потік (Thread) для обробки. Багатопотоковість частково рятує ситуацію, але створення сотень потоків під час ранкового пікового навантаження (коли всі студенти одночасно перевіряють розклад) нещадно з'їдає оперативну пам'ять сервера і змушує процесор витрачати ресурси на постійне перемикання контексту через обмеження Global Interpreter Lock (GIL) у Python.

Щоб зробити систему по-справжньому стійкою до навантажень і здатною витримувати "шторм повідомлень" без зависань, імплементується асинхронна модель обробки запитів.

В основі цього підходу лежить модуль asyncio та концепція циклу подій (Event Loop). Замість того, щоб плодити важкі системні потоки, асинхронна програма працює в одному потоці, але максимально ефективно жонглює завданнями (корутинами). Коли наш бот отримує запит від першого студента і відправляє запит до PostgreSQL, він не чекає відповіді в режимі простою. Завдяки ключовому слову await, функція тимчасово призупиняється і віддає управління назад до Event Loop. Цикл подій бачить, що є вільний час, і миттєво підхоплює повідомлення від наступного користувача. Як тільки дані для першого студента надходять з бази, корутина "прокидається" і продовжує роботу з того ж місця.

З точки зору інструментарію, екосистема Python пропонує два шляхи для реалізації цього механізму:

1. Використання повністю асинхронного фреймворку aiogram. Він створювався з нуля під asyncio, тому кожна його складова працює без блокувань. Це індустріальний стандарт для високонавантажених ботів, де всі обробники оголошуються як async def, а відправка повідомлень відбувається через await message.answer().

2. Використання асинхронної обгортки AsyncTeleBot з бібліотеки telebot. Цей підхід дозволяє зберегти звичний синтаксис декораторів @bot.message_handler, але переводить роботу мережевого шару на асинхронні рейки (через бібліотеку aiohttp), що робить міграцію з синхронного коду менш болючою.

Перехід на асинхронну парадигму вимагає комплексного рефакторингу всієї системи. Недостатньо просто зробити функції бота асинхронними – потрібно, щоб весь ланцюжок викликів був неблокуючим. Саме тому стандартний драйвер бази даних psycopg2, який є синхронним і блокує виконання коду на рівні ОС, у такій архітектурі замінюється на його асинхронні аналоги (наприклад, asyncpg або використання ORM SQLAlchemy з модулем asyncio).

Реалізація асинхронної обробки перетворює Telegram-бота з повільного, послідовного скрипта на високопродуктивний сервіс. Він здатен обробляти тисячі конкурентних з'єднань одночасно, маючи мінімальне споживання оперативної пам'яті (memory footprint). Користувачі отримують блискавичний відгук інтерфейсу (до десятків мілісекунд), що створює ефект безшовного спілкування з живим помічником, а серверні потужності утилізуються максимально ефективно.

3.6 Обробка виняткових ситуацій та помилок введення

Розробка інтерактивних програмних систем, що взаємодіють з користувачем через текстовий інтерфейс вільного вводу, вимагає імплементації надійних механізмів обробки помилок. Оскільки користувач може вводити дані в неправильному форматі, допускати друкарські помилки або намагатися виконати логічно суперечливі

дії, Telegram-бот повинен бути максимально стійким до таких аномалій. Замість використання класичного механізму генерації винятків (Exceptions), який може призвести до аварійного завершення процесу або зупинки обробки запитів інших користувачів, у системі реалізовано патерн безпечного повернення результату (Result Wrapper Pattern).

Для реалізації цього підходу створено спеціалізовані класи-обгортки SubjectParseResult та SubjectRemovalParseResult. Вони інкапсулюють у собі як успішний результат виконання операції (згенерований об'єкт дисципліни), так і опис можливої помилки (поле error). Це дозволяє маршрутизатору повідомлень безпечно перевіряти наявність помилки (if parse_result.error is not None) та миттєво повертати користувачеві відповідне текстове попередження без переривання роботи основного циклу програми.

Процес валідації вхідних даних у системі розділено на три логічні рівні: синтаксичний, семантичний та рівень бізнес-логіки.

На синтаксичному рівні первинна перевірка здійснюється за допомогою регулярних виразів у модулі парсера (шаблони subject_regex та subject_removal_regex). Якщо введений текст взагалі не відповідає очікуваному патерну (наприклад, відсутній час, день тижня або неправильно розставлені пробіли), парсер одразу повертає помилку формату, сигналізуючи про неможливість розпізнати запит.

На семантичному рівні відбувається перевірка логічної цілісності розпізнаних даних, що пройшли синтаксичний контроль:

1. Валідація часу: функція parse_time суворо контролює межі значень (години від 0 до 23, хвилини від 0 до 59). Додатково в парсері перевіряється консистентність часового проміжку: час початку пари математично не може бути більшим за час її закінчення (start > finish), а також вони не можуть дорівнювати один одному (start == finish). У разі виявлення таких парадоксів користувач отримує відповідну підказку.

2. Валідація днів тижня: функція parse_day підтримує гнучкість вводу, розуміючи різні формати (абревіатури "пн", повні назви "понеділок" або цифрові екві-

валенти "1"). Однак, якщо користувач вводить невідоме значення, системі не загрожує помилка виконання типу KeyError. Замість цього повертається безпечний стан model.Day.UNRECOGNIZED, який система коректно обробляє, блокуючи запис до бази даних.

На рівні бізнес-логіки обробляються виняткові ситуації, пов'язані зі станом бази даних та життєвим циклом розкладу:

1. Конфлікти розкладу (Накладання часових слотів): При спробі додати нову пару система в обов'язковому порядку викликає метод chat_repository.find_intersection_slot. Якщо новий часовий проміжок перетинається (має спільні хвилини) з уже існуючою дисципліною у цей самий день для даного чату, запис блокується. Програма формує динамічне повідомлення, вказуючи користувачеві точну назву та час предмета, з яким виник конфлікт.

2. Відсутність даних: При спробі видалити предмет за його позицією, система аналізує кількість фактично змінених рядків у БД (row_count < 1). Якщо предмет під таким номером не знайдено, операція скасовується з відповідним сповіщенням. Аналогічний підхід застосовується до команд перегляду розкладу (/current_day, /week): якщо SQL-запит повертає порожній масив (занять немає), бот не виводить порожнє повідомлення чи помилку Telegram API, а надсилає заздалегідь підготовлений шаблонний текст про відсутність запланованих пар.

Такий комплексний, багаторівневий підхід до валідації даних та обробки виняткових ситуацій гарантує високу стабільність серверної частини бота, унеможливлює потрапляння пошкоджених даних до реляційної бази та формує інтуїтивно зрозумілий користувацький досвід, чітко пояснюючи причину будь-якої відмови системи.

3.7 Аналіз результатів та тестування функціональності

Щоб переконатися у надійності розробленого Telegram-бота для управління розкладом, ми провели комплексне тестування. Воно включало перевірку базового

функціоналу, тестування обробки некоректних даних (negative testing) та аналіз роботи скінченного автомата (FSM), який керує станами чату.

Перевірка базових сценаріїв Ми тестували бота, імітуючи реальні дії користувача. Основні результати перевірок:

Ініціалізація та базові команди: Коли користувач вперше надсилає /start, бот коректно створює запис у БД (через chat_repository.init) і встановлює для чату початковий стан model.State.START. Команда /clear також працює без збоїв – вона успішно видаляє всі предмети, прив'язані до конкретного chat_id, і надсилає повідомлення про підтвердження.

Логіка поточного часу (/now): Ми перевірили, як алгоритм визначає поточну пару. Він бере системний час, переводить його у хвилини і робить запит до бази. Бот без проблем розрізняє три сценарії: інформує про пару, яка йде прямо зараз; показує наступну пару (якщо поточний час менший за час початку заняття); або повідомляє, що на сьогодні пар більше немає.

Виведення розкладу: Команди /current_day, /next_day та /week працюють як слід. Щоб розклад не виводився хаотично (на випадок, якщо користувач додавав предмети не по порядку), ми застосували функцію сортування utility.subject_comparator. Вона гарантує, що масив об'єктів Subject завжди виводиться хронологічно.

Тестування системи управління станами та валідації Окремо тестували те, як бот реагує на введення даних та перемикає контекст.

Переходи між станами: Після виклику команди /add_subject бот слухняно переходить у режим очікування даних (стан TIME_SLOT). Увесь наступний текст маршрутизатор направляє безпосередньо у функцію parse_subject. Аналогічно, /remove_subject правильно перемикає систему на очікування номера пари (стан SUBJECT_POSITION).

Негативне тестування (обробка помилок): Ми спеціально "годували" парсер некоректними даними. Наприклад, якщо написати звичайний текст на кшталт "Додай математику на завтра", який не підпадає під регулярний вираз subject_regex, програма не падає. Вона просто генерує помилку prompt.unable_to_parse_time_slot_message.

Якщо ввести логічно неправильний час (наприклад, 25:00-26:00 або 09:00-08:00), функція parse_time це перехоплює і повертає об'єкт SubjectParseResult із текстом помилки.

Неіснуючі дні тижня також відловлюються: функція parse_day повертає model.Day.UNRECOGNIZED, що одразу блокує спробу запису в БД.

Перевірка бізнес-логіки та роботи з базою даних Одне з найважливіших завдань полягало в захисті бази від логічних колізій.

Захист від накладання пар: Змоделювали ситуацію: користувач намагається додати пару на час, який частково або повністю перетинається з уже існуючою (наприклад, є пара о 08:00-09:30, а ми додаємо на 09:00-10:30). Метод chat_repository.find_intersection_slot успішно знаходить цей конфлікт ще на рівні SQL-запиту. Бот скасовує дію і попереджає користувача. Таким чином, у базі не виникає "каші" з розкладу.

Видалення неіснуючого: Що буде, якщо спробувати видалити 5-ту пару, коли в розкладі їх усього дві? Після виконання DELETE-запиту база повертає row_count = 0. Бот розуміє, що нічого не змінилося, не видає системних помилок, а просто повідомляє користувача, що такої пари не знайдено (prompt.subject_position_was_not_found).

Підсумки Тестування показало, що бот працює стабільно. Обрана архітектура з чіткими моделями (Subject, Event, ChatState) та використанням обгорток для результатів повністю себе виправдала. Жоден із тестів із "битим" або неправильним вводом не призвів до зависання Event Loop чи критичного збою (crash).

Додаток швидко обробляє текст та надійно взаємодіє з PostgreSQL. Скінченний автомат працює без збоїв, стани різних користувачів ізольовані і не перетинаються між собою. Отже, модуль управління розкладом повністю відповідає поставленим вимогам і готовий до використання в реальних умовах.

ВИСНОВКИ

Проведено аналіз обраної предметної галузі в сфері навчання криптографічним методам. Визначено основні потреби користувачів у зручному та інтерактивному освітньому середовищі.

Проведено аналіз програмного забезпечення, що існує для вивчення криптографічних методів. Визначено основні функції, переваги та недоліки наявних рішень, які були враховані при розробці навчального модуля.

Досліджено сучасні підходи до гейміфікації освітнього процесу. Визначено ефективні механіки гейміфікації, що були реалізовані у створеному модулі для підвищення мотивації користувачів.

Визначено функціональні та нефункціональні вимоги до навчального модуля. Проведено моделювання за допомогою діаграм варіантів використання.

Проведено аналіз засобів програмної реалізації. Обрано відповідні інструменти та технології для реалізації вебсайту згідно з поставленими вимогами.

Розроблено основні компоненти навчального модулю, включаючи модуль завдань, елементи гейміфікації та інтерфейс вебсайту, відповідно до визначених вимог.

Реалізовано функціонал створення акаунту користувача, що дозволяє зберігати прогрес навчання та персоналізувати взаємодію з платформою. Перевірка розробленого продукту підтвердила стабільну роботу кожного модуля. Криптографічні алгоритми виконуються без помилок, а дані користувачів зберігаються надійно. Система працює злагоджено та без збоїв.

Сформовані методичні рекомендації дозволяють легко впровадити платформу в освітній процес. Модуль підходить і як інструмент для самостійної роботи, і як база для практичних занять з інформаційної безпеки. Це повністю готове рішення, що не потребує складних додаткових налаштувань.

Створений навчальний модуль є цілісним продуктом, де сувора логіка криптографії поєднується з ігровими елементами. Усі поставлені завдання виконано: інтерфейс зрозумілий, а швидкість роботи висока. Складні алгоритми стають

доступнішими, що підвищує інтерес до навчання. Це створює надійний фундамент для подальшого розширення платформи, наприклад, через додавання нових типів шифрів або проведення змагань у реальному часі. Результати роботи мають практичне значення для цифровізації сучасної освіти, роблячи процес вивчення складних тем ефективним та динамічним.

ПЕРЕЛІК ПОСИЛАНЬ

1. Zelle, J. Python Programming: An Introduction to Computer Science / J. Zelle. – Franklin, Beedle & Associates Inc., 2010. – 510 с.

2. Sweigart, A. Automate the Boring Stuff with Python / A. Sweigart. – No Starch Press, 2015. – 504 с.

3. Grinberg, M. Flask Web Development: Developing Web Applications with Python / M. Grinberg. – O'Reilly Media, 2018. – 316 с.

4. Masinter, L., & Hardt, D. SQLAlchemy: Database Access Using Python / L. Masinter, D. Hardt. – O'Reilly Media, 2015. – 304 с.

5. Brownlee, J. Mastering REST APIs in Python: Using Flask-Restful, Flask-SQLAlchemy, and Flask-JWT-Extended / J. Brownlee. – Machine Learning Mastery, 2018. – 300 с.

6. Smith, A. Building Telegram Bots: Develop Bots in 12 Programming Languages Using the Telegram Bot API / A. Smith. – Independently published, 2019. – 200 с.

7. Richardson, L., & Ruby, S. RESTful Web Services / L. Richardson, S. Ruby. – O'Reilly Media, 2007. – 448 с.

8. Timofeev, I. Telegram Bot Development for Beginners / I. Timofeev. – CreateSpace Independent Publishing Platform, 2018. – 234 с.

9. Lutz, M. Learning Python, 5th Edition / M. Lutz. – O'Reilly Media, 2013. – 1648 с.

10. Alchin, M. Pro Python / M. Alchin. – Apress, 2010. – 368 с.

11. Box, G., & Draper, N. R. Empirical Model-Building and Response Surfaces / G. Box, N. R. Draper. – Wiley, 1987. – 688 с.

12. Ronacher, A. Flask Documentation.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
import psycopg2
import telebot
import functools

import chat_repository
import mapper
import message_parser
import model
import prompt
import startup
import utility
import datetime

bot=
telebot.TeleBot("6945395938:AyEUn6z9wLN72PBct3LAч
NTYdYPE9RVYguc")
conn = psycopg2.connect(database='diplom',
user='postgres', host='localhost', password=' password ',
port= port)
startup.create_schema_structure(conn)
@bot.message_handler(commands=["start"])
def send_welcome(message):
    event = mapper.to_event(message)
    chat_repository.init(conn, event.chat_id)

@bot.message_handler(commands=["help"])
def print_help(message):
    event = mapper.to_event(message)
    chat_repository.update_state(conn, event.chat_id,
model.State.START)
    bot.send_message(event.chat_id, prompt.help,
parse_mode="markdown")
@bot.message_handler(commands=["clear"])
def clear_schedule(message):
    event = mapper.to_event(message)
    chat_repository.clear_schedule(conn, event.chat_id)
    chat_repository.update_state(conn, event.chat_id,
model.State.START)
    bot.send_message(event.chat_id,
prompt.successful_schedule_clearing,
parse_mode="markdown")

@bot.message_handler(commands=["add_subject"])
def add_subject(message):
    event = mapper.to_event(message)
    chat_repository.update_state(conn, event.chat_id,
model.State.TIME_SLOT)
    bot.send_message(event.chat_id,
prompt.type_time_slot_message,
parse_mode="markdown")

@bot.message_handler(commands=["remove_subject"])
def remove_subject(message):
    event = mapper.to_event(message)
    chat_repository.update_state(conn, event.chat_id,
model.State.SUBJECT_POSITION)
    bot.send_message(event.chat_id,
prompt.type_subject_position, parse_mode="markdown")
@bot.message_handler(commands=["current_day"])
def current_day(message):
    event = mapper.to_event(message)
    chat_repository.update_state(conn, event.chat_id,
model.State.START)
    now = datetime.datetime.now()
    weekday = now.weekday()
    current_day = model.days[weekday]
    subjects =
chat_repository.get_schedule_by_chat_id_and_day(conn,
event.chat_id, current_day)
    if len(subjects) == 0:
        bot.send_message(event.chat_id,
prompt.empty_current_day, parse_mode="markdown")
        return
    subjects = sorted(subjects,
key=functools.cmp_to_key(utility.subject_comparator))
    message =
utility.compose_daily_schedule_message(current_day,
subjects)
    bot.send_message(event.chat_id, message,
parse_mode="markdown")

@bot.message_handler(commands=["next_day"])
def next_day(message):
    event = mapper.to_event(message)
    chat_repository.update_state(conn, event.chat_id,
model.State.START)
    now = datetime.datetime.now()
    weekday = now.weekday()
    next_day = model.days[(weekday + 1) % 7]
    subjects =
chat_repository.get_schedule_by_chat_id_and_day(conn,
event.chat_id, next_day)
    if len(subjects) == 0:
        bot.send_message(event.chat_id,
prompt.empty_current_day, parse_mode="markdown")
        return
    subjects = sorted(subjects,
key=functools.cmp_to_key(utility.subject_comparator))
    message =
utility.compose_daily_schedule_message(next_day,
subjects)
    bot.send_message(event.chat_id, message,
parse_mode="markdown")

@bot.message_handler(commands=["week"])
def week(message):
    event = mapper.to_event(message)
    chat_repository.update_state(conn, event.chat_id,
model.State.START)
    subjects =
chat_repository.get_weekly_schedule_by_chat_id(conn,
event.chat_id)
    if len(subjects) == 0:
        bot.send_message(event.chat_id, prompt.empty_week,
parse_mode="markdown")
```

```
        return
    week = dict()
    for subject in subjects:
        if subject.day not in week:
            week[subject.day] = []
        week[subject.day].append(subject)
    message =
utility.compose_weekly_schedule_message(week)
    bot.send_message(event.chat_id, message,
parse_mode="markdown")

@bot.message_handler(commands=["now"])
def now(message):
    event = mapper.to_event(message)
    now = datetime.datetime.now()
    weekday = now.weekday()
    minutes = now.hour * 60 + now.minute
    current_day = model.days[weekday]
    subject = chat_repository.get_current_subject(conn,
event.chat_id, current_day, minutes)
    if subject is None:
        bot.send_message(event.chat_id,
prompt.no_subjects_for_today, parse_mode="markdown")
        return
    if subject.start > minutes:
        bot.send_message(event.chat_id, "*Наступний
предмет*\n" +
                         utility.time_to_string(subject.start) + "-" +
utility.time_to_string(subject.finish) + ": " +
                         subject.title, parse_mode="markdown")
        return
    bot.send_message(event.chat_id, "*На даний час
проходить*\n" +
                     utility.time_to_string(subject.start) + "-" +
utility.time_to_string(subject.finish) + ": " +
                     subject.title, parse_mode="markdown")

@bot.message_handler(content_types=["text"])
def process_message(message):
    event = mapper.to_event(message)
    state_record = chat_repository.get_chat_state(conn,
event.chat_id)
    if state_record.state == model.State.TIME_SLOT:
        parse_result =
message_parser.parse_subject(event.text)
        if parse_result.error is not None:
            bot.send_message(event.chat_id, parse_result.error)
            return
        subject = parse_result.subject
        intersection = chat_repository.find_intersection_slot(
            conn,
            event.chat_id,
            subject.day,
            subject.start,
            subject.finish)
        if intersection is not None:
            bot.send_message(
                event.chat_id,
                "Неможливо додати даний предмет у розклад,
оскільки він конфліктує з предметом \"" +
intersection.subject + "\", що проходить о " +
utility.time_to_string(
                    intersection.start) + "-" +
utility.time_to_string(intersection.finish),
                parse_mode="markdown")
            return
        chat_repository.insert_schedule(conn, event.chat_id,
parse_result.subject)
        bot.send_message(event.chat_id,
prompt.successful_subject_addition,
parse_mode="markdown")
    elif state_record.state ==
model.State.SUBJECT_POSITION:
        parse_result =
message_parser.parse_subject_removal(event.text)
        if parse_result.error is not None:
            bot.send_message(event.chat_id, parse_result.error,
parse_mode="markdown")
            return
        row_count =
chat_repository.delete_by_chat_id_and_day_and_position(
            conn,
            event.chat_id,
            parse_result.day,
            parse_result.position)
        if row_count < 1:
            bot.send_message(event.chat_id,
prompt.subject_position_was_not_found)
            return
        bot.send_message(event.chat_id,
prompt.successful_subject_removal)

bot.infinity_polling()

import model

def to_event(message):
    return model.Event(
        message_id=message.id,
        from_id=message.from_user.id,
        from_first_name=message.from_user.first_name,
        from_last_name=message.from_user.last_name,
        from_username=message.from_user.username,
        chat_id=message.chat.id,
        text=message.json["text"])

import re

import model
import prompt

subject_regex = r"^\s*([^\s]*)\s*([0-9]{1,2}):([0-9]{1,2})[-
\s]+([0-9]{1,2}):([0-9]{1,2})\s+(.+)$"
subject_removal_regex = r"^\s*([^\s]*)\s+([0-9]+)\s*$"

def parse_subject(text):
    regex_result = re.match(subject_regex, text)
    if regex_result:
        day = parse_day(regex_result.group(1))
        if day == model.Day.UNRECOGNIZED:
            return model.SubjectParseResult(None,
prompt.unable_to_parse_day_message)
        raw_start = (regex_result.group(2),
regex_result.group(3))
        start = parse_time(raw_start[0], raw_start[1])
        if start < 0:
            return model.SubjectParseResult(None,
prompt.unable_to_parse_start_time_message)
        raw_finish = (regex_result.group(4),
regex_result.group(5))
        finish = parse_time(raw_finish[0], raw_finish[1])
        if finish < 0:
            return model.SubjectParseResult(None,
prompt.unable_to_parse_finish_time_message)
```

```
        if start > finish:
            return model.SubjectParseResult(None,
prompt.start_time_is_larger_than_to_time_error_message)
        if start == finish:
            return model.SubjectParseResult(None,
prompt.equal_from_time_and_to_time_error_message)
        title = regex_result.group(6).strip()
        return model.SubjectParseResult(
            subject=model.Subject(
                day=day,
                title=title,
                start=start,
                finish=finish),
            error=None)
    else:
        return model.SubjectParseResult(None,
prompt.unable_to_parse_time_slot_message)

def parse_subject_removal(text):
    regex_result = re.match(subject_removal_regex, text)
    if regex_result:
        day = parse_day(regex_result.group(1))
        if day == model.Day.UNRECOGNIZED:
            return model.SubjectRemovalParseResult(None,
None, prompt.unable_to_parse_day_message)
        position = int(regex_result.group(2))
        if position < 1:
            return model.SubjectRemovalParseResult(None,
None, prompt.invalid_position_error_message)
        return model.SubjectRemovalParseResult(day,
position, None)
    else:
        return model.SubjectRemovalParseResult(None,
None,
prompt.unable_to_parse_subject_remove_slot_message)

def parse_time(raw_hours, raw_minutes):
    hours = int(raw_hours)
    minutes = int(raw_minutes)
    if hours < 0 or hours > 23 or minutes < 0 or minutes >
59:
        return -1
    return hours * 60 + minutes

def parse_day(raw_day):
    raw_day = raw_day.lower()
    if raw_day == "пн" or raw_day == "понеділок" or
raw_day == "1":
        return model.Day.MONDAY
    elif raw_day == "вт" or raw_day == "вівторок" or
raw_day == "2":
        return model.Day.TUESDAY
    elif raw_day == "ср" or raw_day == "середа" or
raw_day == "3":
        return model.Day.WEDNESDAY
    elif raw_day == "чт" or raw_day == "четвер" or
raw_day == "4":
        return model.Day.THURSDAY
    elif raw_day == "пт" or raw_day == "п'ятниця" or
raw_day == "пятниця" or raw_day == "5":
        return model.Day.FRIDAY
    elif raw_day == "сб" or raw_day == "субота" or
raw_day == "6":
        return model.Day.SATURDAY
    elif raw_day == "нд" or raw_day == "неділя" or
raw_day == "7":
        return model.Day.SUNDAY
    return model.Day.UNRECOGNIZED

from enum import Enum

class Day(Enum):
    MONDAY = "MONDAY"
    TUESDAY = "TUESDAY"
    WEDNESDAY = "WEDNESDAY"
    THURSDAY = "THURSDAY"
    FRIDAY = "FRIDAY"
    SATURDAY = "SATURDAY"
    SUNDAY = "SUNDAY"
    UNRECOGNIZED = "UNRECOGNIZED"

class State(Enum):
    START = "START"
    TIME_SLOT = "TIME_SLOT"
    SUBJECT_POSITION = "SUBJECT_POSITION"

days = list(Day)

class Event:
    def __init__(self, message_id, from_id, from_first_name,
from_last_name, from_username, chat_id, text):
        self.message_id = message_id
        self.from_id = from_id
        self.from_first_name = from_first_name
        self.from_last_name = from_last_name
        self.from_username = from_username
        self.chat_id = chat_id
        self.text = text

class ChatState:
    def __init__(self, id, chat_id, state):
        self.id = id
        self.chat_id = chat_id
        self.state = state

class Subject:
    def __init__(self, day, title, start, finish):
        self.day = day
        self.title = title
        self.start = start
        self.finish = finish

class SubjectParseResult:
    def __init__(self, subject, error):
        self.subject = subject
        self.error = error

class SubjectRemovalParseResult:
    def __init__(self, day, position, error):
        self.day = day
        self.position = position
        self.error = error

class TimeSlot:
    def __init__(self, subject, start, finish):
        self.subject = subject
        self.start = start
```

```
false);
    return;
  }
  const correctAnswer = caesarCipher(text, key, ALPHABET_UA,false);
  if (userAnswer === correctAnswer) { showFeedback('feedback-3', `Правильно!
+${POINTS_PER_TASK} балів!`, true); updateScore(POINTS_PER_TASK); markSectionComplete(2);
    document.getElementById('check-decrypt-
btn').disabled = true; document.getElementById('decrypt-btn').disabled =
true;
    document.getElementById('decrypt-result').disabled
= true;
    setTimeout(()=>showSection(4),  1000);
  } else {
    showFeedback('feedback-3', `Неправильно, правильна відповідь: ${correctAnswer}`, false);
  }
});

document.getElementById('next-btn- 3').addEventListener('click', () => {
  // більше не потрібно
});
```

```
function updateProgress() {
  const percent = ((parseInt(section)-1)/4)*100; document.getElementById('progress-bar').style.width = percent
+ '%';
}
function nextSection(next) { document.getElementById('section-' + section).style.display
= 'none'; document.getElementById('section-' + section).classList.remove('visible');
  if (section === '1') setVijenerAchievement('vijener_intro'); if (section === '1-5')
setVijenerAchievement('vijener_rules');
  section = next;
  document.getElementById('section-' + section).style.display
= '';
  setTimeout(()=>{ document.getElementById('section-' +
section).classList.add('visible');
    // Анімація появи фіналу
    if (section === '5') {
      const final = document.querySelector('.dmc-final'); if(final) final.classList.add('visible');
      setVijenerAchievement('vijener_all_done');
    }
  }, 10);
  updateProgress();
}
let encryptTries = 0;
function isValidLAText(text) {
  // Дозволяємо лише латинські літери, пробіли, розділові знаки
  return /^[A-Z .,!?\-'"\n\r]+$/i.test(text);
```