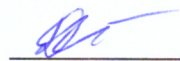


Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи магістра
на тему: «МЕТОДИ ТА ТЕХНОЛОГІЇ ОБМІНУ
ДАНИМИ З ВИКОРИСТАННЯМ БЕЗПЕЧНИХ АРІ»

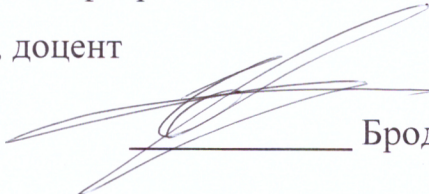
Виконав студент 2-го курсу групи КБм-23-1
спеціальності 125 «Кібербезпека та захист інформації»

 Гусєв В.С.

Керівник доцент кафедри КІ та КБ, кандидат
технічних наук, доцент

 Маєвський О.В.

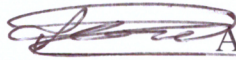
Рецензент доцент кафедри КІ та КБ, кандидат
технічних наук, доцент

 Бродський Ю.Б.

Житомир – 2024

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 125 «Кібербезпека та захист інформації»
Освітня програма «Кібербезпека»

ЗАТВЕРДЖУЮ

Завідувач
кафедри комп'ютерної
інженерії та кібербезпеки
 Андрій Єфіменко

1 лютого 2024 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу магістра

Студент: Гусев Віталій Сергійович

Тема роботи: «Методи та технології обміну даними з використанням
безпечних API»

затверджена Наказом університету від 1 лютого 2024 р. № 479/с

Термін здачі студентом закінченої роботи 30 грудня 2024 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

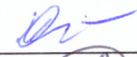
Методи та технології розробки настільних застосунків на Python з
використанням бібліотек PyQt5/PySide6 та API.

Обладнання та програмне забезпечення:

1. Python 3.11 – основна мова програмування для розробки застосунку.
2. Visual Studio Code – середовище розробки з необхідними розширеннями:
3. Python (офіційне розширення для підтримки мови).
4. Pylance (інструмент для автодоповнення коду).
5. Code Runner (для швидкого запуску скриптів).
6. PyQt5/PySide6 – бібліотеки для створення графічного інтерфейсу програми.
7. Requests – бібліотека для інтеграції зовнішніх API та обробки HTTP-запитів.
8. OpenWeather API – для отримання актуальної інформації про погоду.
9. WorldTime API – для відображення часу у різних часових поясах.
10. НБУ API – для отримання курсів валют.
11. REST Countries API – для надання інформації про країни.
12. Google Translate API – для реалізації функціоналу перекладу текстів.
13. Git – система контролю версій для збереження та управління вихідним кодом.
14. SQLite – для збереження даних користувача у локальній базі даних.

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	О.О. Шелуха	01.11.2024	01.11.2024
2	О.О. Шелуха	16.11.2024	16.11.2024
3	О.О. Шелуха	03.12.2024	03.12.2024

Студент  Гусев В.С.

Керівник  Маєвський О.В.

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	01.11.2024 – 07.11.2024	Виконано
2	Розробка плану роботи	07.11.2024 – 14.11.2024	Виконано
3	Огляд та аналіз доступних API для інтеграції функціоналу програми	14.11.2024 – 16.11.2024	Виконано
4	Налаштування середовища розробки та бібліотек для створення графічного інтерфейсу	16.11.2024 – 21.11.2024	Виконано
5	Розробка системи авторизації для захисту даних користувачів	21.11.2024 – 03.12.2024	Виконано
6	Тестування функціонування програми	05.12.2024 – 12.12.2024	Виконано
7	Написання пояснювальної записки	12.12.2024 – 16.12.2024	Виконано
8	Захист	30.12.2024	Виконано

Студент  Гусєв В.С.

Керівник  Маєвський О.В.

АНОТАЦІЯ

У даній роботі представлено розробку багатофункціонального настільного застосунку з використанням мови програмування Python та бібліотек PyQt5/PySide6 для створення графічного інтерфейсу. Основна мета проєкту – надання користувачам зручного та інтуїтивно зрозумілого інтерфейсу для вирішення повсякденних завдань через інтеграцію з відкритими API.

Випускна кваліфікаційна робота магістра складається з пояснювальної записки, що містить вступ, три розділи, висновки, список використаної літератури та додатки. Текстова частина викладена на 59 сторінках друкованого тексту. Пояснювальна записка має 18 сторінок додатків. Список використаних джерел містить 16 найменувань і займає 2 сторінки. В роботі наведено 12 рисунків. Загальний обсяг роботи – 80 сторінок.

Функціонал програми включає:

- Систему авторизації, що забезпечує безпеку даних користувача.
- Годинник із часовими поясами, який отримує дані через WorldTime API.
- Калькулятор валют, що виконує конвертацію на основі даних НБУ API.
- Календар і погодні модуль, що відображають глобальні погодні умови та інформацію про поточний день за допомогою OpenWeather API.
- Інформаційний модуль про країни, який надає дані про столицю, населення, валюту та прапор за допомогою REST Countries API.
- Перекладач, що використовує API для швидкого перекладу текстів між обраними мовами.

У роботі детально описано процес налаштування середовища розробки Visual Studio Code для ефективної роботи з Python та інтеграції необхідних бібліотек і API.

Ключові слова: PYTHON, PyQt5, PySide6, API, VS CODE, АВТОРИЗАЦІЯ, КОНВЕРТАЦІЯ ВАЛЮТ, ПОГОДНИЙ МОДУЛЬ, ГРАФІЧНИЙ ІНТЕРФЕЙС.

ANNOTATION

This work presents the development of a multifunctional desktop application using the Python programming language and PyQt5/PySide6 libraries to create a graphical interface. The main goal of the project is to provide users with a convenient and intuitive interface for solving everyday tasks through integration with open APIs.

The final qualification work of the master consists of an explanatory note, including an introduction, three chapters, conclusions, a list of references, and appendices. The test part is set out on 59 pages of printed text. The explanatory note includes 18 pages of appendices. The list of references contains 16 sources and occupies 2 pages. The work contains 12 figures. The total volume of the work is 80 pages.

The functionality of the program includes:

- An authorization system that ensures the security of user data.
- A clock with time zones that receives data via the WorldTime API.
- A currency calculator that performs conversion based on NBU API data.
- A calendar and weather module that displays global weather conditions and information about the current day using the OpenWeather API.
- A country information module that provides data about the capital, population, currency, and flag using the REST Countries API.
- A translator that uses the API to quickly translate texts between selected languages.

The paper describes in detail the process of setting up the Visual Studio Code development environment for effective work with Python and the integration of the necessary libraries and APIs.

Keywords: Python, PyQt5, PySide6, API, VS Code, AUTHORIZATION, CURRENCY CONVERSION, WEATHER MODULE, GRAPHICAL INTERFACE.

Зміст

ВСТУП	9
РОЗДІЛ I. ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО МЕТОДИ ТА ТЕХНОЛОГІЇ API.....	11
1.1. Методи та технології обміну даними через API	11
1.2. Веб-сервіси: функціональність, дані та стандарти.....	21
1.3. SOAP: історія, особливості та застосування	24
1.4. WSDL: ключ до взаємодії з веб-сервісами.....	27
Висновок до розділу 1	38
РОЗДІЛ II. Тестування API	39
2.1. Тестування застосунків, що надають API	39
2.2. Порівняння тестування API з модульним тестуванням (JUnit).....	41
2.3. Особливості тестування API	41
2.4. Тестування веб-сервісів	43
Висновок до розділу 2	46
РОЗДІЛ III. СЕРЕДОВИЩЕ РОЗРОБКИ ТА ІНТЕРФЕЙС ПРОГРАМИ.	47
3.1. Середовище розробки VS Code	47
3.2. Налаштування VS code	49
3.3. Огляд власної розробки.....	53
Висновок до розділу 3	58
ВИСНОВОК.....	59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	60
ДОДАТКИ.....	62

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API — Програмний інтерфейс взаємодії, що дозволяє обмінюватися даними між застосунками або сервісами.

PyQt5/PySide6 — Бібліотеки для створення графічних інтерфейсів користувача на мові програмування Python.

VS Code — Інтегроване середовище розробки (IDE) для написання, редагування та налагодження коду.

HTTP/HTTPS — Протоколи передачі гіпертексту, що використовуються для обміну інформацією в мережі.

ВСТУП

У сучасному світі, де інформація є важливим ресурсом, програмні рішення відіграють ключову роль у спрощенні повсякденних завдань. Створення настільних застосунків з інтуїтивно зрозумілим інтерфейсом та широким функціоналом є актуальним завданням для підвищення продуктивності користувачів.

Актуальність теми. Сучасний розвиток інформаційних технологій сприяє створенню багатофункціональних програмних продуктів, які забезпечують користувачів необхідною інформацією в реальному часі та спрощують виконання повсякденних завдань. Одним із важливих напрямків є розробка настільних застосунків, що поєднують роботу з відкритими API, обробку даних та інтуїтивно зрозумілий графічний інтерфейс. Невирішеними залишаються питання ефективної інтеграції даних з різних джерел, розробки безпечних систем авторизації та забезпечення зручності використання програмного забезпечення. Результати досліджень сучасних вчених та практика роботи з відкритими API показують актуальність таких рішень для широкого застосування у різних сферах діяльності, як-от фінансовий аналіз, планування часу, обробка текстової інформації та моніторинг погодних умов.

Мета роботи. Розробка багатофункціонального настільного застосунку з використанням мови програмування Python та бібліотек PyQt5/PySide6, що забезпечує інтуїтивно зрозумілий графічний інтерфейс для виконання завдань на основі інтеграції з відкритими API.

Завдання роботи:

Для досягнення мети роботи необхідно вирішити наступні завдання:

1. Провести аналіз можливостей середовища розробки Visual Studio Code для роботи з Python.

2. Виконати огляд та аналіз доступних API для інтеграції функціоналу програми (погода, курси валют, часові пояси, інформація про країни, переклад текстів).

3. Налаштувати середовище розробки та необхідні бібліотеки для створення графічного інтерфейсу.

4. Розробити систему авторизації для захисту даних користувачів.

Реалізувати функціонал програми, що включає: годинник із підтримкою часових поясів; калькулятор валют; модуль погоди та календар; інформаційний блок про країни; перекладач текстів. Виконати тестування функціоналу програми та оцінити її продуктивність.

Об'єкт дослідження. Процес інтеграції зовнішніх API для отримання та обробки даних у реальному часі в настільних застосунках.

Предмет дослідження. Методи та технології розробки настільних застосунків на Python з використанням бібліотек PyQt5/PySide6 та API.

Методи дослідження. У роботі використовуються методи проєктування програмних систем, об'єктно-орієнтоване програмування, а також методи роботи з API для обробки зовнішніх даних. Застосовано методи тестування для перевірки працездатності програмного продукту.

Практичне значення отриманих результатів. Розроблений програмний продукт може бути використаний у повсякденній діяльності для отримання інформації про час, погоду, курси валют та країни, а також для швидкого перекладу текстів. Програма є корисною як для індивідуальних користувачів, так і для підприємств, що працюють з глобальною інформацією.

РОЗДІЛ I. ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО МЕТОДИ ТА ТЕХНОЛОГІЇ API

API (Application Programming Interface) — це програмний інтерфейс, який дозволяє взаємодіяти між різними програмами, сервісами або системами. Безпечний API забезпечує обмін даними, який захищає конфіденційність, цілісність інформації та контролює доступ до ресурсів. Він є важливою частиною сучасних технологій, особливо в середовищах, де передача чутливих даних є критичною (банкінг, охорона здоров'я, електронна комерція тощо).

1.1. Методи та технології обміну даними через API

Методи обміну даними через API. Це способи, якими дві системи обмінюються даними через API:

1. Запити та відповіді: Клієнт (застосунок) надсилає запит до сервера через API, а сервер повертає відповідь із даними.

- Використовуються протоколи, такі як HTTP/HTTPS.
- Дані передаються у форматах JSON, XML, або інших.

2. Виклик процедур: Реалізується через віддалений виклик процедур (RPC) або сучасні RESTful API. Це дозволяє викликати методи сервера, отримуючи результат.

3. Потоки даних: API може забезпечувати постійний потік даних у реальному часі, наприклад, через WebSocket або Server-Sent Events (SSE).

Технології обміну даними через API. Це набір інструментів, протоколів і стандартів, які забезпечують захищений обмін даними між системами. Основні технології включають:

1. REST (Representational State Transfer)

- Найпоширеніший підхід для створення веб-сервісів.
- Використовує HTTP-методи (GET, POST, PUT, DELETE) для обміну даними.
- Дані часто передаються у форматі JSON або XML.

2. SOAP (Simple Object Access Protocol)

- Більш формалізований підхід для передачі даних через XML.
- Використовується у складних корпоративних системах із високими вимогами до безпеки.

3. gRPC

- Використовує протокол HTTP/2 і формат даних Protocol Buffers для швидкої та ефективної передачі даних.
- Підходить для створення високопродуктивних розподілених систем.

4. WebSocket

- Протокол для двостороннього зв'язку в реальному часі між клієнтом і сервером.
- Використовується для чату, біржових даних або інших сервісів із швидкою зміною інформації.

5. GraphQL

- Гнучкий підхід, який дозволяє клієнтам отримувати лише потрібні їм дані.
- Підходить для складних додатків із великими наборами даних.

Важливість використання безпечного API.

Безпечні API захищають обмін даними між клієнтом і сервером від:

1. Несанкціонованого доступу: Забезпечують, що лише автентифіковані та авторизовані користувачі можуть отримувати дані.
2. Витоку даних: Використовують шифрування для запобігання перехопленню даних.
3. Атак типу "людина посередині" (MITM): Забезпечують безпеку каналів зв'язку.

Основні компоненти безпеки в API:

1. HTTPS: Захищає передачу даних між клієнтом і сервером.
2. OAuth 2.0 та OpenID Connect: Забезпечують безпечну авторизацію.
3. JWT (JSON Web Tokens): Використовуються для передачі інформації про автентифікацію та авторизацію.
4. Rate Limiting: Захищає API від перевантаження або атак DDoS.
5. Шифрування: Забезпечує безпеку чутливих даних.

API також є засобом інтеграції застосунків. Зовнішній програмний інтерфейс (іноді інтерфейс прикладного програмування) (англ. Application Programming Interface, API) — це набір готових класів, процедур, функцій, структур та констант, які надаються застосунком (бібліотекою, сервісом) для використання у зовнішніх програмних продуктах.

API визначає функціональність, яку надає програма (модуль, бібліотека), при цьому API дозволяє абстрагуватися від того, як саме ця функціональність реалізована.

Зовнішній програмний інтерфейс є системою уніфікованих зв'язків, призначених для обміну даними між компонентами системи. API задає набір

необхідних процедур, їхніх параметрів та способів звернення, а також надає функціональність, яку певний програмний компонент надає іншим програмним компонентам.

Зворотній зв'язок як характеристика управління.

Зворотній зв'язок — це одна з характеристик управління. У складних системах зворотній зв'язок може розглядатися як передача інформації про хід процесу, на основі якої виробляється те чи інше керуюче впливання. У цьому випадку зворотній зв'язок називають інформаційним.

За наявності зворотного зв'язку результат попередньої дії впливає на подальший хід процесу, а самокерована система, завдяки наявності в ній прямих і зворотних зв'язків (тобто керуючої та коригувальної інформації), функціонує за замкнутим циклом. Цей досить стійкий і добре структурований цикл лежить в основі управлінського циклу і називається контуром управління.

Контур управління — це найпростіша абстрактна модель управління. Вона розглядає управління як інформаційний процес і показує рух інформації в системі. Поняття контуру управління лежить в основі циклічного характеру управління.

Циклічність управління пов'язана з постійною взаємодією керуючої та керованої підсистем через канали прямого і зворотного інформаційного зв'язку, що типово для складних функціональних систем. Перетворення інформації зворотного зв'язку в інформацію управлінських рішень, по суті, і є змістом процесу управління.

Система із зворотним зв'язком, попри свою простоту, використовується практично в будь-якій системі управління. Основна проблема полягає у її правильному налаштуванні, щоб вона не заважала, а допомагала у реалізації проєкту.

API як абстракція функціональності та реалізації компонентів.

API може розглядатися як абстракція над функціональністю та реалізацією компонентів. Іншими словами, API посиляється на набір функцій, вбудованих у застосунки, які можуть бути використані іншими застосунками (або самим застосунком, як ми побачимо пізніше), щоб взаємодіяти із застосунком.

API є чудовим способом надійно й безпечно розкрити функціональність застосунку для зовнішніх програм. Усі функціональні можливості, доступні зовнішнім застосункам, обмежуються наданим API.

API як інструмент для надання доступу до функціональності.

API дозволяє додаткам взаємодіяти через стандартизовані протоколи та механізми. Це дає змогу зовнішнім програмам використовувати лише ті функції, які чітко визначені та доступні через API, забезпечуючи захищений доступ до функціональності.

Завдяки цьому API стає важливим інструментом для інтеграції систем, оскільки дозволяє абстрагуватися від технічних деталей реалізації та зосередитися на взаємодії між компонентами. Наприклад, зовнішній застосунок може отримати доступ до даних, не переймаючись, як вони зберігаються або обробляються.

Системи управління і взаємодія через API.

Складні системи, які включають різноманітні модулі та підсистеми, часто використовують API для забезпечення взаємодії між компонентами. Завдяки цьому можна досягти гнучкості та модульності системи, що особливо важливо для великих проєктів.

Зворотній зв'язок у таких системах сприяє оптимізації процесів. Інформація про стан системи передається через API для подальшого аналізу та прийняття управлінських рішень. Це дозволяє створити замкнуті цикли управління, які забезпечують безперервний моніторинг і коригування роботи системи.

API у практичному використанні.

API широко використовується в різних галузях:

1. *Веб-розробка:* API дозволяють взаємодіяти між веб-сервісами, наприклад, отримувати дані про погоду, карти або обробляти платежі.
2. *Мобільні додатки:* мобільні застосунки часто використовують API для доступу до серверів, баз даних або сторонніх сервісів.
3. *Автоматизація:* API використовується для налаштування сценаріїв автоматизації між різними платформами.

У кожному з цих випадків API виступає універсальним інструментом, який забезпечує надійну, структуровану та безпечну взаємодію між програмами.

API системи "KSU FEEDBACK".

API системи "KSU FEEDBACK" є інструментом, за допомогою якого інформація, накопичена системою веб-аналітики, стає доступною для використання на стороні іншого сайту. Цей механізм відкриває широкі можливості для програмістів і адміністраторів сайтів. Наприклад, можна перенаправити відвідувача на потрібну сторінку залежно від його статусу і прав, надаючи йому таким чином унікальну інформацію.

Реалізація механізму API. API реалізовано у вигляді набору функцій, які можуть використовуватися на сайті для отримання даних із сервера аналітики.

Для використання функцій API необхідно вставити код віджета на сторінку сайту. Цей код стає доступним власнику сайту після реєстрації в системі " ". Таким чином, API дозволяє налаштовувати зв'язки між різними застосунками для ефективної взаємодії.

API також добре підходить для поширення застосунків через інші сайти, дозволяючи інтегрувати функціональність у зовнішні ресурси.

Особливості WebAPI. У веб-розробці зазвичай використовується визначений набір HTTP-запитів, а також структура HTTP-відповідей, для яких застосовуються формати XML або JSON. WebAPI практично є синонімом веб-служби, хоча останнім часом спостерігається перехід від SOAP до REST-типу комунікації.

Гібридні веб-інтерфейси. Веб-інтерфейси, що забезпечують поєднання декількох сервісів у нових застосунках, відомі як гібридні. Вони використовуються для створення інноваційних рішень, об'єднуючи можливості кількох API у єдиній системі.

Це забезпечує високу гнучкість і масштабованість у розробці сучасних веб-застосунків.

Сервіси, що надають API

Багато сучасних сервісів, які досягли значного масштабу й отримали популярність, відкривають свої API для сторонніх застосунків. Це дозволяє інтегрувати їхні функції у зовнішні системи та створювати нові можливості.

Приклади сервісів з API:

1. Twitter

Twitter надає широкі можливості для сторонніх застосунків:

- Отримання інформації про конкретний обліковий запис.
- Пошук у базі даних сервісу.
- Аутентифікація користувачів через Twitter.

На базі Twitter API створено величезну кількість сторонніх сайтів і застосунків. Також багато сервісів використовують Twitter-аутентифікацію як свою основну.

2. Feedburner. API цього сервісу дозволяє:

- Отримувати інформацію про RSS-канал вибраного блогу.
- Управляти налаштуваннями RSS-каналу.

API-орієнтовані веб-застосунки.

API-орієнтоване веб-застосунок — це застосунок, у якому весь або більша частина функціоналу реалізується через виклики API.

Особливості API-орієнтованих застосунків:

- Передача даних через API: Наприклад, для авторизації користувач надсилає свої дані через функцію API. Результатом виконання може бути успішна авторизація або повідомлення про помилку.
- Незалежність від стану користувача: API не залежить від поточного стану користувача. Це обмеження змушує розробників створювати чисту функціональність, що полегшує тестування й забезпечує зручність використання.

Переваги API-орієнтованих веб-застосунків:

1. *Кросплатформність*: Функціональність, реалізована через API, може використовуватися будь-яким пристроєм — браузером, мобільним телефоном, планшетом або настільним застосунком.

2. *Централізоване управління:* API дозволяє створити централізоване застосунок, який прийматиме виклики від будь-якого пристрою користувача.

3. *Масштабованість:* Завдяки стандартизованим викликам API легко інтегрувати нові пристрої чи платформи без значних змін у коді.

Створення API-орієнтованих застосунків дозволяє розробникам забезпечувати гнучкість, зручність і функціональність, що відповідає сучасним вимогам до програмного забезпечення.

Організація роботи API веб-служби:

SOAP

Для ефективного використання веб-сервісів і технології SOAP (Simple Object Access Protocol) необхідно розуміти принципи та технології, на яких вони базуються.

Що таке веб-сервіс? Веб-сервіс — це інтерфейс для функціональності застосунку, доступний через мережу, побудований із використанням стандартних інтернет-технологій. Якщо застосунок може отримати доступ до функціоналу через мережу за допомогою комбінації протоколів (наприклад, HTTP, XML, SMTP), то він є веб-сервісом.

Принципи роботи API:

API веб-сервісу зазвичай складається з двох основних типів функцій:

1. *Функції, що повертають дані:* Сторонній застосунок надсилає запит до сервера з конкретними параметрами, а у відповідь отримує необхідну інформацію у визначеному форматі (наприклад, JSON або XML).

2. *Функції, що змінюють дані:* Застосунок-клієнт викликає функцію сервера, яка змінює певні налаштування або додає нову інформацію.

Особливості методів HTTP:

- *GET-запити:* Використовуються для отримання даних. Важливо, щоб виклик через GET не змінював стан сервера (наприклад, не оновлював дані в базі або лог-файли). Це забезпечує можливість кешування запитів і знижує навантаження на сервер.
- *POST/PUT-запити:* Використовуються для зміни або додавання даних. Якщо потрібно змінити стан сервера, це обов'язково має бути реалізовано через POST або PUT.

Переваги SOAP у веб-сервісах.

SOAP розв'язує проблеми кросплатформної взаємодії застосунків. До його основних переваг належать:

- *Простота налаштування:* SOAP заснований на XML, що дозволяє створювати універсальні повідомлення для передачі даних.
- *Кросплатформність:* SOAP працює через різні транспортні протоколи, включаючи HTTP і SMTP, забезпечуючи взаємодію між системами.
- *Безпека:* SOAP забезпечує підтримку захищених з'єднань, включаючи передачу даних через брандмауери.

Основні етапи роботи з API (на прикладі Twitter API):

1. Надсилення GET-запиту з необхідними параметрами до адреси потрібної функції.
2. Отримання у відповідь даних у форматі JSON або XML.
3. Якщо потрібно — обробка (парсинг) отриманих даних.

SOAP забезпечує стандартизований спосіб обміну даними між застосунками, що особливо важливо для розподілених систем. Завдяки цьому протоколу стало можливим ефективно вирішувати задачі взаємодії між різними платформами.

1.2. Веб-сервіси: функціональність, дані та стандарти

Веб-сервіси — це програмні компоненти, що надають функціональність і дані для використання зовнішніми застосунками. Взаємодія з веб-сервісами здійснюється за допомогою стандартних протоколів (наприклад, HTTP) і форматів даних (наприклад, XML або JSON). Веб-сервіси є незалежними від мови програмування і платформи реалізації, що робить їх універсальним інструментом для інтеграції.

Роль XML у веб-сервісах. XML (eXtensible Markup Language) є стандартом для опису, структурування та передачі даних у веб-сервісах. Його ключові характеристики:

- *Управління даними:* XML дозволяє програмно обробляти інформацію за допомогою визначень, таких як DTD або схеми.
- *Універсальність:* Теги в XML можуть бути узгоджені між сторонами, інтерфейси визначені, а обробка даних стандартизована.
- *Інтеграція бізнес-логіки:* XML-документи можуть містити логіку автоматизації, транзакцій, упорядкування компонентів, що дозволяє інтегрувати бізнес-процеси у корпоративні застосунки.

Особливості веб-сервісів

1. *Компонентні програми багаторазового використання:* Веб-сервіси розробляються для багаторазового застосування у різних бізнес-сценаріях.
2. *Використання стандартних протоколів:* HTTP виступає основним транспортним протоколом для передачі даних і бізнес-логіки через веб-сервіси.
3. *Доступ до корпоративних даних:* Веб-сервіси забезпечують інтерфейси для доступу до великого обсягу корпоративної інформації, що зберігається у традиційних репозиторіях.

Переваги використання веб-сервісів:

- *Інтеграція:* XML допомагає інтегрувати веб-дані з корпоративними застосунками, координацією бізнес-логіки та уніфікацією процесів.
- *Автоматизація:* Автоматизовані транзакції збільшують продуктивність, знижують витрати та покращують якість послуг.
- *Гнучкість:* Веб-сервіси спрощують роботу з бізнес-даними та забезпечують зв'язок між компонентами системи, підтримуючи ланцюгову взаємодію в рамках Інтернету.

Важливість стандартів у веб-сервісах.

Присутність стандартів, таких як XML і HTTP, дозволяє автоматизувати транзакції між застосунками. Це значно підвищує продуктивність і знижує складність роботи з розподіленими системами.

Веб-сервіси відкривають нові можливості для підприємств, дозволяючи використовувати дані й бізнес-логіку в нових сценаріях, оптимізуючи бізнес-процеси й створюючи конкурентні переваги.

Використання веб-сервісів для покращення бізнес-процесів.

Автоматизація та її вплив на ефективність. Однією з ключових переваг веб-сервісів є автоматизація транзакцій, яка:

- Підвищує продуктивність за рахунок усунення ручної обробки даних.
- Знижує витрати завдяки стандартизації процесів і використанню відкритих протоколів.
- Покращує якість послуг через мінімізацію людських помилок і швидкий доступ до необхідних даних.

Автоматизація стає можливою завдяки використанню стандартних форматів даних (XML, JSON) і транспортних протоколів (HTTP), які спрощують обмін інформацією між системами.

XML як основа інтеграції. XML виконує центральну роль у веб-сервісах, дозволяючи:

- Створювати універсальну структуру для передачі даних, яка легко читається та аналізується.
- Інтегрувати існуючі бізнес-процеси та логіку в нові системи, що розширює можливості підприємств.
- Координувати компоненти різних систем, об'єднуючи їх у єдине середовище для виконання складних бізнес-операцій.

Цінність слабозв'язаних веб-сервісів. Слабозв'язані веб-сервіси створюють просту й гнучку структуру для інтеграції, що забезпечує:

- Мінімальну залежність між компонентами, що полегшує масштабування систем.

- Швидке оновлення та зміну окремих частин сервісу без впливу на загальну архітектуру.
- Можливість використовувати різні технології, платформи та мови програмування в межах однієї системи.

Вплив на підприємства. Веб-сервіси дозволяють підприємствам:

- Покращувати взаємодію між внутрішніми та зовнішніми системами.
- Пропонувати свої послуги іншим компаніям або кінцевим користувачам через відкриті API.
- Оптимізувати бізнес-процеси, використовуючи автоматизовані транзакції та централізовані сервіси.

Приклади застосування веб-сервісів:

1. Електронна комерція: Інтеграція платіжних систем через API дозволяє обробляти транзакції в режимі реального часу.
2. Логістика: Веб-сервіси використовуються для моніторингу доставки, відстеження товарів і оптимізації маршрутів.
3. Фінанси: Банки та фінансові установи інтегрують свої сервіси для обміну даними про рахунки, транзакції та курси валют.

1.3. SOAP: історія, особливості та застосування

Виникнення SOAP.

SOAP (Simple Object Access Protocol) — це технологія, що виникла на основі раннього XML-стандарту, відомого як XML-RPC. У певному сенсі, SOAP прагне до більш розширеного стандарту ebXML (Electronic Business XML), який спрямований на створення універсальних бізнес-повідомлень між

торговими партнерами. Водночас SOAP є менш складним і більш легким у впровадженні, ніж ebXML.

Веб-сервіси та слабке зв'язування. Веб-сервіси відокремлюють об'єкти від платформ, які їх пов'язують. А це означає, що вони:

- Полегшують взаємодію між об'єктами, незалежно від їхньої платформи.
- Забезпечують доступ до даних з будь-якої точки у веб-просторі.

Веб-сервіси використовують слабкі зв'язки між компонентами, що робить їх більш стійкими до змін у системі. У той час як сильно зв'язані системи залежать від специфічних для однієї платформи об'єктів, слабо зв'язані системи є більш універсальними та гнучкими.

Проблеми сильно зв'язаних систем:

- Залежність від платформи: якщо система спирається на унікальний інтерфейс, будь-яка зміна або помилка може призвести до збоїв.
- Крихкість: розрив у з'єднанні між компонентами може викликати непередбачувані наслідки.

Переваги слабо зв'язаних систем.

Системи з слабкими зв'язками, наприклад, система KSU Feedback, мають такі переваги:

- *Модульність*: Поділ системи на окремі модулі, які взаємодіють через програмні інтерфейси.
- *Стабільність*: Система може продовжувати працювати навіть у разі виявлення помилок у окремих модулях.

- *Гнучкість*: Простота адаптації та оновлення окремих компонентів без впливу на всю систему.

Використання SOAP у веб-сервісах:

SOAP дозволяє створювати слабко зв'язані системи, оскільки:

- Використовує XML для структурування даних, забезпечуючи незалежність від платформи.
- Забезпечує передачу даних через стандартні транспортні протоколи (наприклад, HTTP, SMTP).
- Підтримує складні бізнес-логіки, такі як транзакції та автоматизація процесів.

Застосування системи «KSUFeedback». Система KSUFeedback є прикладом слабко зв'язаної архітектури завдяки:

- Поділу на модулі.
- Наявності чітко визначених програмних інтерфейсів.

Це забезпечує стабільну роботу навіть у разі помилок або виняткових ситуацій, а також сприяє масштабованості та зручності підтримки.

Веб-сервіси, що базуються на SOAP, ідеально підходять для побудови слабко зв'язаних систем, забезпечуючи гнучкість, стабільність і простоту інтеграції в складних корпоративних середовищах.

Структура веб-сервісів: цикл «публікувати-знаходити-зв'язувати»

Структура веб-сервісів базується на циклі «публікувати-знаходити-зв'язувати», який забезпечує взаємодію між постачальниками послуг та запитуючими сторонами:

1. *Публікація*: Постачальники роблять доступними свої дані, контент або послуги.
2. *Пошук*: Запитуючі сторони локалізують необхідні сервіси через реєстраційні механізми.
3. *Зв'язування*: Встановлюється з'єднання між сервісом і запитуючим застосунком.

1.4. **WSDL: ключ до взаємодії з веб-сервісами**

WSDL (Web Services Description Language) — це мова опису веб-сервісів, яка дозволяє застосункам взаємодіяти з ними, забезпечуючи:

- *Опис інтерфейсів*: Надає низькорівневу технічну інформацію про сервіси, включаючи методи, доступні для виклику, і їхні параметри.
- *Кодировку даних*: Використовує XML Schema для забезпечення правильного форматування даних.
- *Підтримку протоколів*: Гарантує виконання правильних операцій за правильними протоколами, такими як HTTP або SOAP.

Абстракція та конкретизація у WSDL. WSDL працює на двох рівнях:

1. *Абстрактний рівень*: Описує абстрактний інтерфейс веб-сервісу, включаючи доступні методи та структуру даних. Це дозволяє використовувати сервіс незалежно від специфіки його реалізації.
2. *Конкретизація*: Зв'язує абстрактний інтерфейс із конкретним транспортним механізмом (наприклад, HTTP) і адресою сервісу.

Таким чином, WSDL виконує функцію універсального опису, що дозволяє повторно використовувати веб-сервіси.

Переваги використання WSDL.

- *Стандартизація:* Забезпечує єдиний формат опису сервісів, що полегшує інтеграцію.
- *Масштабованість:* Дозволяє легко адаптувати сервіси до нових протоколів і середовищ.
- *Автоматизація:* Дає змогу застосункам автоматично зчитувати опис сервісу та взаємодіяти з ним без ручного налаштування.

Структура веб-сервісів складається з циклу «публікувати-знаходити-зв'язувати», за допомогою якого постачальники сервісів роблять дані, контент або послуги доступними для зареєстрованих сторінок, що надсилають запити, знаходячи сервіси та підключаючись до них.

Запитуючі застосунки налаштовуються на веб-сервіси за допомогою WSDL (Web Services Definition Language — мова опису веб-сервісів), який надає низькорівневу технічну інформацію про бажаний сервіс, дозволяє звертатися до XML Schema для кодування даних і гарантує виконання правильних операцій через відповідні протоколи.

Однак WSDL є не тільки мовою для опису інтерфейсів, він також включає конструкції, які дозволяють описувати інформацію про адресу і протокол опублікованого веб-сервісу.

Цікаво, що WSDL описує абстрактний інтерфейс для веб-сервісу, одночасно дозволяючи (у найменших деталях) зв'язувати веб-сервіс із конкретним транспортним механізмом, таким як HTTP. Абстрагуючи інтерфейс, WSDL функціонує як технологія багаторазово використовуваних веб-сервісів.

Зв'язуючи веб-сервіс із конкретним транспортним механізмом, WSDL робить абстракцію конкретною.

SOAP дозволяє створювати застосунки, викликаючи методи об'єктів віддалено. SOAP усуває вимогу, щоб дві системи працювали на одній платформі або були написані однією мовою програмування.

Замість виклику методів через певний бінарний протокол, пакет SOAP використовує XML — текстовий синтаксис для виконання викликів методів. Уся інформація між застосунком-запитувачем і приймаючим об'єктом передається у вигляді даних, укладених між тегами в XML-поточці через HTTP.

З точки зору веб-сервісів, SOAP може бути реалізований як клієнт, так і сервер.

Для надання функціональності системи «KSUFeedback» стороннім застосункам необхідно було визначити XMLSchema для об'єктів, з якими буде працювати API. Це потрібно для подальшої роботи з повідомленнями SOAPClient і перетворення відповідей сервера у форматі XML для подальшого відображення на стороні користувача.

Механізм взаємодії клієнта і сервера

1. Клієнтський застосунок створює екземпляр об'єкта SOAPClient.
2. SOAPClient читає файли опису методів веб-сервісу (WSDL і WebServicesMetaLanguage — WSMML). Ці файли можуть зберігатися і на стороні клієнта.
3. Клієнтський застосунок, використовуючи можливості пізнього зв'язування методів об'єкта SOAPClient, викликає метод сервісу. SOAPClient формує пакет запиту (SOAPEnvelope) і відправляє його на сервер. Може використовуватися будь-який транспортний протокол, але зазвичай використовується HTTP.
4. Пакет приймає серверне застосування Listener, створює об'єкт SOAPServer і передає йому пакет запиту.

5. SOAPServer читає опис веб-сервісу, завантажує опис і пакет запиту в XMLDOM-дерева.
6. SOAPServer викликає метод об'єкта/застосунку, що реалізує сервіс.
7. Результати виконання методу або опис помилки конвертуються об'єктом SOAPServer у пакет відповіді та відправляються клієнту.
8. Об'єкт SOAPClient обробляє прийнятий пакет і повертає клієнтському застосунку результати роботи сервісу або опис виниклої помилки.

Файл WSDL — це документ у форматі XML, який описує методи, що надаються веб-сервісом. Також він містить інформацію про параметри методів, їхні типи, назви та місцезнаходження Listener'а сервісу. Майстер SOAPToolkit автоматично генерує такий документ.

SOAP Envelope (Пакет) — це XML-документ, який містить запит або відповідь на виконання методу. Зручніше за все уявляти його як поштовий конверт, у який вкладена інформація. Тег *Envelope* повинен бути кореневим елементом пакета.

- Елемент *Header* не є обов'язковим.
- Елемент *Body* обов'язково має бути присутнім і бути прямим нащадком елемента *Envelope*.

У разі помилки виконання методу сервер формує пакет, який у тегі *Body* містить елемент *Fault*, де надається детальний опис помилки.

Якщо ви використовуєте високорівневі інтерфейси SOAPClient і SOAPServer, вам не потрібно заглиблюватися у тонкощі формату пакета. Однак, якщо є бажання, можна скористатися низькорівневими інтерфейсами або навіть створити пакет вручну.

Об'єктна модель SOAPToolkit

SOAPToolkit дозволяє працювати з об'єктами низькорівневого API:

- *SoapConnector* — Забезпечує роботу з транспортним протоколом для обміну SOAP-пакетами.
- *SoapConnectorFactory* — Надає метод створення конектора для транспортного протоколу, зазначеного у WSDL-файлі (тег).
- *SoapReader* — Читає SOAP-повідомлення та будує XMLDOM-дерева.
- *SoapSerializer* — Містить методи для створення SOAP-повідомлень.
- *IsoapTypeMapper*, *SoapTypeMapperFactory* — Інтерфейси, які дозволяють працювати зі складними типами даних.

Робота з типами даних.

Використовуючи об'єкти високорівневого API, можна передавати дані тільки простих типів (наприклад, *int*, *string*, *float*). Однак специфікація SOAP 1.1 дозволяє працювати зі складними типами даних, такими як:

- Масиви,
- Структури,
- Списки,
- Комбінації цих типів.

Для роботи з такими типами даних необхідно використовувати інтерфейси *IsoapTypeMapper* і *SoapTypeMapperFactory*.

Клієнт SOAP — це програма, яка створює XML-документ, що містить інформацію, необхідну для віддаленого виклику методу у розподіленій системі.

Окрім того, що клієнти SOAP розширюють різноманітність застосунків, вони також можуть використовуватися як веб-серверні програми або програми, засновані на серверних технологіях. Повідомлення і запити від клієнтів SOAP зазвичай передаються через HTTP.

У результаті документи SOAP здатні обійти майже будь-який брандмауер (firewall), забезпечуючи обмін інформацією між різними платформами.

Сервер SOAP — це спеціальний код, який прослуховує повідомлення SOAP і діє як розподільник і інтерпретатор документів SOAP.

Зовнішні веб-сервіси можуть взаємодіяти із застосунками-серверами, заснованими на технології J2EE, які обробляють запити SOAP від багатьох клієнтів.

Сервери SOAP гарантують, що документи, отримані через HTTP-з'єднання, перетворені у мову, зрозумілу об'єкту і всьому його оточенню.

Оскільки всі комунікації здійснюються у формі XML, об'єкти, написані однією мовою (наприклад, Java), можуть взаємодіяти через SOAP з об'єктами, написаними іншою мовою (наприклад, C++).

Робота SOAP-сервера полягає в тому, щоб переконатися, що кінцеві точки розуміють SOAP-запити, які їх обслуговують.

SOAP є розширюваним, оскільки клієнти SOAP, сервери та сам протокол можуть розвиватися, не руйнуючи вже існуючі застосунки.

Формат повідомлень. Усі вищезазначені процеси виконуються у контексті стандартизованого формату повідомлень. Основна частина повідомлення має MIME-тип text/xml і містить SOAP-конверт. Цей конверт є XML-документом.

SOAP-конверт містить:

- Заголовок (опціонально).
- Тіло (обов'язково).

Тіло конверта завжди призначене кінцевому одержувачу повідомлення, тоді як записи у заголовку можуть бути адресовані вузлам, які виконують проміжну обробку. До тіла також можуть бути додані бінарні або інші вкладення.

SOAP дозволяє клієнту визначити, який із проміжних вузлів обробляє ту чи іншу запис у заголовку. Оскільки заголовки є незалежними від основного вмісту SOAP-повідомлення, вони корисні для додавання до повідомлення інформації, яка не впливає на обробку його тіла.

Наприклад, заголовки можуть бути використані для додавання цифрового підпису до запиту, розміщеного в тілі повідомлення. У цьому випадку сервер автентифікації або авторизації може обробити запис у заголовку (незалежно від тіла), розбираючи інформацію для перевірки правильності підпису. Після успішної перевірки решта конверта передається на SOAP-сервер, який обробляє тіло повідомлення.

Більш детальний аналіз SOAP-конверта допомагає зрозуміти розташування та мету заголовка і елементів тіла.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

```

<SOAP-ENV:Header>
  <t:Transaction xmlns:t="some-URI" SOAP-ENV:mustUnderstand="1">
    5
  </t:Transaction>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <m:GetPollBySyrveyID xmlns:m="some-URI">
    <ID>700</ID>
  </m:GetPollBySyrveyID>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

У цьому прикладі запит *GetPollBySyrveyID* (отримати анкету за ідентифікатором опитування) надсилається службі через веб.

- *Запит приймає параметр типу строка (ідентифікатор).*
- *Функція повертає об'єкт «Анкета» у форматі відповіді SOAP.*
- *SOAP-конверт — це верхній елемент XML-документа, який представляє повідомлення SOAP.*

- *Простори імен XML використовуються для відокремлення ідентифікаторів SOAP від специфічних ідентифікаторів застосунку. Основна мета просторів імен XML — кваліфікувати елементи в повідомленні або віднести їх до певної області.*

Щоб зрозуміти, як працюють простори імен SOAP, необхідно ознайомитися зі специфікацією просторів імен XML. Простір імен можна уявити як ідентифікатор області елементів, який допомагає унікально ідентифікувати елементи SOAP, асоціюючи їх із конкретним місцем.

- **Тіло повідомлення SOAP** у прикладі містить корисне навантаження XML, яке, припустимо, виконує RPC (Remote Procedure Calling — виклик віддаленої процедури). SOAP — це не тільки модульна, але й досить прихована модель організації пакетів.

- У документі немає нічого, що явно вказує на те, що був виконаний RPC. Все, що ми бачимо в тілі, — це пара XML-елементів, один із яких кваліфікований простором імен.

- **Завдання SOAP-сервера.** SOAP-сервер відповідає за розуміння семантики документа і виконання правильних дій. Насправді сервер надає структуру для роботи з корисним навантаженням XML (XML payload) «значущим» способом.

- У цьому контексті слово «значущий» означає, що сервер виконує виклик віддаленої процедури у певній допоміжній базі даних, щоб отримати об'єкт, використовуючи параметр, який міститься в тілі повідомлення.

Повідомлення SOAP по суті є односторонньою передачею інформації від відправника до отримувача, однак повідомлення SOAP часто об'єднуються для поліпшення механізмів запиту і відповіді.

Для реалізації RPC у SOAP необхідно дотримуватися певних угод:

1. Повідомлення запиту і відповіді повинні бути закодовані як структури.
2. Для кожного вхідного параметра операції повинен існувати елемент з таким самим іменем, як у параметра.
3. Для кожного вихідного параметра повинен існувати елемент з відповідною назвою.

Приклад SOAP-RPC повідомлень. У роботі буде наведено скорочений вигляд SOAP-RPC повідомлень, що складаються лише з тіл запитів і відповідей.

Запит (Request)

```
xml
<SOAP-ENV:Body>
  <m:GetPollBySurveyID xmlns:m="some-URI">
    <ID>700</ID>
  </m:GetPollBySurveyID>
</SOAP-ENV:Body>
```

Відповідь (Response)

```
xml
<SOAP-ENV:Body>
  <m:GetPollBySurveyIDResponse xmlns:m="some-URI">
    <price>22.50</price>
  </m:GetPollBySurveyIDResponse>
</SOAP-ENV:Body>
```

У цьому прикладі:

- *Запит* передає параметр ID із значенням 700.
- *Відповідь* повертає елемент price зі значенням 22.50.

Запит викликає метод *GetPollBySurveyID*. Відповідь визначає операцію *GetPollBySurveyIDResponse*.

Загальна угода SOAP вимагає, щоб наприкінці операції запиту (*Request*) додавалося слово *Response* для створення структури відповіді (*Response*). Ця вихідна структура містить елемент із назвою *ID* (ідентифікатор), який повертає результат виклику методу, можливо, у вигляді об'єкта з його властивостями (поллями).

Також важливо зазначити, що у SOAP-конверті типи даних ніде не виражені явно, тому, лише дивлячись на повідомлення SOAP, ми дійсно не знаємо тип ID або тип результуючого параметра об'єкта (наприклад, анкети).

Клієнтські застосунки визначають типи даних:

- Відкрито через кодування.
- Або неявно через узгоджені контакти із сервером.

У будь-якому випадку ці визначення не включені явно в SOAP-повідомлення.

Протоколи для RPC у SOAP. Для реалізації RPC потрібен низькорівневий транспортний протокол, такий як HTTP.

- Специфікація SOAP 1.0 передбачає використання HTTP як основного транспортного протоколу.
- Специфікація SOAP 1.1 (і «SOAP-повідомлення з вкладеннями») дозволяє також використовувати інші протоколи:

- FTP
- SMTP
- TCP/IP-сокети (за можливості).

Усі правила серіалізації та кодування SOAP також застосовуються до RPC-параметрів.

Висновок до розділу 1

WSDL (Web Services Description Language) є ключовою технологією для опису інтерфейсів веб-сервісів, що забезпечує їхню стандартизацію, автоматизацію та інтеграцію у різних програмних середовищах. Завдяки розділенню на абстрактний рівень і конкретизацію, WSDL дозволяє описувати доступні методи, параметри та структури даних, одночасно пов'язуючи їх із конкретними транспортними механізмами, такими як HTTP чи SOAP. Це робить веб-сервіси універсальними та багаторазово використовуваними.

SOAP, у свою чергу, забезпечує ефективний механізм обміну XML-повідомленнями між клієнтами й серверами, дозволяючи виклик методів у розподілених системах. Завдяки підтримці різних транспортних протоколів і незалежності від платформ чи мов програмування, SOAP стає гнучким інструментом для реалізації RPC і обміну даними.

API (Application Programming Interface) є інтерфейсом для взаємодії між програмами та сервісами, що надає набір методів і правил для отримання або передачі даних. У контексті веб-сервісів API дозволяє клієнтським застосункам надсилати запити до серверів, обробляти відповіді та інтегрувати функціональність зовнішніх сервісів у свої системи. Завдяки WSDL API набуває чіткої структури та формату, що спрощує його використання і забезпечує ефективну взаємодію.

Таким чином, WSDL, SOAP і API створюють комплексний механізм для інтеграції, автоматизації та обміну даними між застосунками в сучасних інформаційних системах.

РОЗДІЛ II. Тестування API

2.1. Тестування застосунків, що надають API

Зовнішній програмний інтерфейс (API) дозволяє програмному забезпеченню взаємодіяти з іншими застосунками. Як і будь-яке інше програмне забезпечення, інтерфейси повинні бути протестовані. Виникає питання: як можна розробити тести без впливу через графічний інтерфейс?

Відповідь: можна використовувати програмні інтерфейси, які забезпечують прямий доступ до функціональності, що підлягає тестуванню.

SoapUI Pro (або QualityLogic) — це кросплатформне рішення для тестування веб-сервісів і API. За допомогою SoapUI Pro можна швидко створювати та запускати автоматизовані функціональні, регресійні та тести безпеки.

Особливості тестування API. Тестування API суттєво відрізняється від інших видів тестування, оскільки графічний інтерфейс (GUI) рідко залучається. Це створює певні труднощі, наприклад:

- Перевірка того, чи система готова до тестування.
- Оцінка функціональності, відповідності вимогам і безпеки.

Задачі тестувальників під час тестування застосунків:

1. *Перевірка викликів API з різними параметрами.* Це необхідно для перевірки функціональності та безпеки. Включає дослідження граничних умов.
2. *Створення комбінацій значень параметрів.* Особливо актуально для викликів із двома або більше параметрами.
3. *Визначення контенту викликів API.* Це може включати:

- Створення зовнішніх умов середовища (файлів, периферійних пристроїв тощо).
- Роботу з внутрішніми даними, що зберігаються, які впливають на API.

4. Послідовність викликів API для перевірки функціональності.

Порядок тестів має відповідати плану тестування. Кожен тест повинен бути максимально автономним і ізольованим від залежностей наскільки це можливо. Основний орієнтир — виявлення найпоширеніших параметрів і умов, які використовуються під час виклику методів API.

Тестування API відрізняється від інших видів тестування тим, що GUI рідко бере участь у процесі. Необхідно ініціалізувати тестовий стан, створивши умови, за яких API буде використовуватися.

Тестові сценарії для API базуються на входних даних:

1. ***Повернене значення на основі умов:*** Надсилаються різні комбінації запитів, а відповіді перевіряються на відповідність очікуваному результату.
2. ***Відсутність поверненого значення:*** Перевіряється поведінка API в системі у разі, коли немає поверненого значення.
3. ***Запуск інших API / подій / виключень:*** Якщо методи API викликають подію чи виключення, вони повинні бути відстежені відповідними обробниками. Набір тестів має перевіряти ці виклики.
4. ***Оновлення даних:*** Оновлення даних може вплинути на систему, тому воно має бути перевірене.
5. ***Зміна певних ресурсів:*** Якщо виклик API змінює ресурси (наприклад, оновлює базу даних), це має бути підтверджено через доступ до відповідних ресурсів.

2.2. Порівняння тестування API з модульним тестуванням (JUnit)

1. *Відмінність у рівні тестування:*

- API-тестування не є модульним тестуванням. Модульним тестуванням зазвичай займаються розробники, тоді як API-тестування виконує команда QA.
- API-тестування — це «чорний ящик», тоді як модульне тестування — це «білий ящик».

2. *Сфери застосування:*

- Обидва види тестування орієнтовані на рівень коду.
- Існує кілька інструментів з відкритим кодом для тестування API, таких як **Webinject**, **JUnit**, **XMLUnit**, **HttpUnit**, **ANT** тощо.

3. *Процес тестування:*

- API-тестування перевіряє методи функціональності.
- Модульне тестування забезпечує розробники для перевірки кожного модуля перед складанням продукту.

4. *Різниця у фокусі:*

- Юніт-тестування перевіряє окремі блоки коду. Взаємодія елементів системи часто не враховується.
- API-тестування спрямоване на функціональний рівень системи, як його використовуватимуть кінцеві користувачі.

Таким чином, API-тестування охоплює ширший спектр, ніж юніт-тестування.

2.3. Особливості тестування API

API-тестування — це спеціальні виклики функцій API з усіма основними варіантами параметрів. Його особливості включають:

- Використання **JMeter** для навантажувального тестування.
- Формування сценаріїв навантаження, характерних для веб-сервісів.
- Зміна параметрів дозволяє відстежити динаміку навантаження під час виклику функцій застосунку користувачами.

Розробка тестів для API

Тестові сценарії повинні бути згруповані за категоріями (граничні випадки, нульові вхідні дані тощо).

Стратегія тестування:

1. **Створення тестів:** Тести пишуться на стандартній мові програмування, яка дозволяє використовувати API-функції.
2. **Виконання тестів:** Необхідне спеціальне середовище для запуску тестів.
3. **Оцінка результатів тестів:** Очікувані результати зазвичай визначаються під час створення тестів.

Атрибути якості:

- **Супровід і підтримка:** Високі. API-функції зазвичай стабільніші, ніж інтерфейс користувача.
- **Перевірка:** Середня. Тести пишуться на тій самій мові програмування і в тому самому середовищі, що й модульні/API-тести, що дозволяє багатьом розробникам розуміти код.

- **Цілісність і залежність:** Середні. Повністю автоматизувати тестування графічного інтерфейсу користувача неможливо; деякі тести все одно потрібно виконувати вручну.
- **Можливість повторного використання:** Низька.

2.4. Тестування веб-сервісів

Веб-сервіси — це поширений компонент розподілених систем. Вони є одним із видів інтерфейсів віддаленого виклику процедур, який дозволяє:

- Надсилати запити на виконання операцій на сервері.
- Отримувати результати виконання без участі графічного інтерфейсу.

Ці компоненти реалізують інтеграцію між системами, які спочатку не були пов'язані. Завдяки цьому веб-сервіси набули популярності, що обумовлює необхідність їх автоматизованого тестування.

Інструменти, такі як **SoapUI** і **TestComplete** (починаючи з 6-ї версії), забезпечують можливість тестування веб-сервісів.

Початкові умови для тестування API

1. **Створення варіантів виклику API:** API може бути викликаний у відповідь на певні події чи винятки.
2. **Обробка помилок у SOAP:** SOAP має механізм обробки помилок, який дозволяє визначати умови виникнення помилки та сповіщати про неї відповідні вузли.

3. *Аналіз загроз*

Загрози для веб-сервісів включають:

- ***Неправильне використання:*** Сервіс повинен перевіряти валідність усіх вхідних і вихідних даних.
- ***Відмова в обслуговуванні:*** Використовується балансування навантаження або розподіл сервісу на різні машини.
- ***Несанкціоноване використання:*** Реалізується через надійну аутентифікацію клієнтів.

Методологія тестування API

- ***Синтаксичне тестування:*** Перевірка функцій API на широкому спектрі вхідних даних.
- ***Функціональне тестування:*** Перевірка коректності роботи окремих функцій або їх комбінацій.
- ***Тестування некоректного застосування:*** Перевірка роботи функцій у непередбачених умовах.
- ***Інтеграційне тестування:*** Перевірка взаємодії API з іншими компонентами або інтерфейсами.

Автоматизація тестування. Модель автоматизації включає:

1. Побудову моделі використання функцій API.
2. Генерацію тестових сценаріїв у XML-форматі.
3. Створення тестового застосунку, що перевіряє функціональність API відповідно до специфікації.

Результати тестування

Тестові сценарії виконуються автоматично, а результат представлений як:

- ***Passed*** — тест пройдено.
- ***Failed*** — тест не пройдено.

Цей підхід підвищує ймовірність виявлення помилок як у роботі окремих функцій, так і у загальній функціональності системи.

Висновок до розділу 2

Тестування веб-сервісів є критично важливим етапом забезпечення їх надійності, функціональності та безпеки. Завдяки автоматизованим інструментам, таким як SoapUI і TestComplete, тестування стає більш структурованим і ефективним. Початкові умови для тестування API передбачають створення сценаріїв виклику, обробку помилок та аналіз загроз, що дозволяє мінімізувати ризики некоректного використання, відмови в обслуговуванні чи несанкціонованого доступу.

Методологія тестування охоплює синтаксичне, функціональне, інтеграційне тестування та перевірку некоректного застосування, що дозволяє детально аналізувати роботу API у різних умовах. Автоматизація тестування забезпечує швидку генерацію тестових сценаріїв і виконання тестів, надаючи чіткі результати у форматі Passed або Failed. Це дозволяє оперативно виявляти помилки, підвищуючи загальну якість та стабільність веб-сервісів у розподілених системах.

Таким чином, тестування веб-сервісів є невідомою частиною циклу розробки API, що забезпечує їх надійність, продуктивність та відповідність специфікаціям.

РОЗДІЛ III. СЕРЕДОВИЩЕ РОЗРОБКИ ТА ІНТЕРФЕЙС ПРОГРАМИ.

3.1. Середовище розробки VS Code

Visual Studio Code (VS Code) є одним із найпопулярніших текстових редакторів та середовищ розробки завдяки своїм численним перевагам. Ось основні причини, чому ми обрали **VS Code** для реалізації нашого проєкту:

1. Легка вага та висока швидкість

VS Code швидко завантажується, має невеликий розмір інсталяції та працює плавно навіть на комп'ютерах з обмеженими ресурсами.

Це ідеально підходить для ефективної та безперебійної роботи над проєктом.

2. Підтримка різних мов програмування

VS Code підтримує багато мов програмування з коробки (наприклад, Python, JavaScript, C++, Java, HTML, CSS).

Завдяки розширенням можна додати підтримку будь-якої мови та синтаксичного підсвічування.

3. Можливість розширення функціоналу

У VS Code є величезна галерея розширень (Extensions Marketplace), які допомагають налаштувати середовище під специфіку проєкту.

Можна встановити плагіни для Git, лінтерів, дебагерів, Docker та інші інструменти.

4. Інтеграція з Git

VS Code має вбудовану інтеграцію з Git, що дозволяє зручно керувати версіями коду прямо з редактора.

Можна комітити, пушити, робити злиття гілок та переглядати зміни без необхідності користуватися терміналом.

5. Інтегрований термінал

Наявність вбудованого терміналу дозволяє виконувати команди безпосередньо у редакторі, що значно підвищує продуктивність роботи.

6. Підтримка відлагодження (Debugging)

У VS Code є потужний вбудований дебагер, що дозволяє виконувати кроковий запуск програм, перевіряти змінні та вирішувати проблеми в коді.

7. Підтримка спільної роботи (Live Share)

За допомогою Live Share кілька розробників можуть одночасно працювати над одним кодом у реальному часі.

8. Безкоштовний та кросплатформний

VS Code є безкоштовним та доступним для Windows, macOS та Linux, що робить його зручним для всіх членів команди.

9. Висока спільнота підтримки

Завдяки величезній спільноті розробників, проблеми, які виникають, легко вирішити за допомогою порад та документації.

10. Гарний інтерфейс і кастомізація

Інтерфейс VS Code є зручним, мінімалістичним і легко налаштовуваним.

Можна встановити теми, змінювати гарячі клавіші та оптимізувати робочий процес.(Рисунок 3.1.)

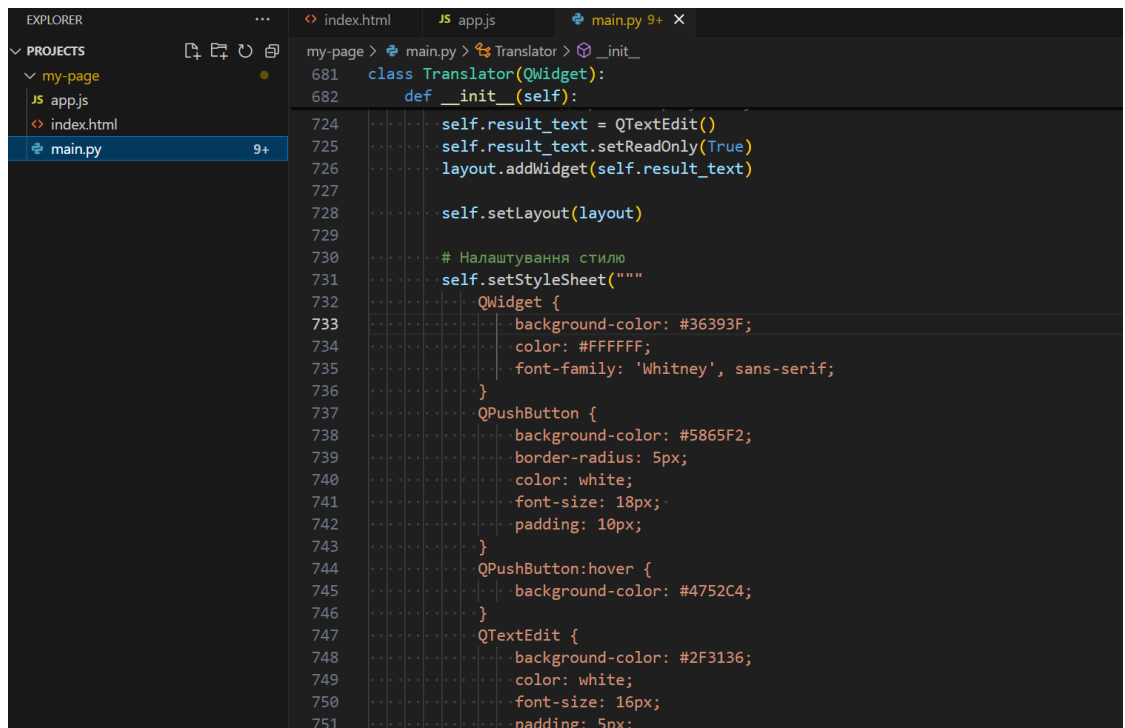


Рисунок 3.1. Зовнішній вигляд Visual Studio Code.

3.2. Налаштування VS code

Щоб налаштувати середовище для Python у Visual Studio Code з використанням PyQt5, PySide6, requests, та інших бібліотек, дотримуйтеся наступних кроків:

1. Встановлення Python

- 1) Завантажте та встановіть Python з [офіційного сайту Python](#).
- 2) При встановленні обов'язково виберіть опцію "Add Python to PATH".(Рисунок 3.2.)
- 3) Перевіряємо, чи Python встановлений:

В терміналі VS Code потрібно прописати
python –version

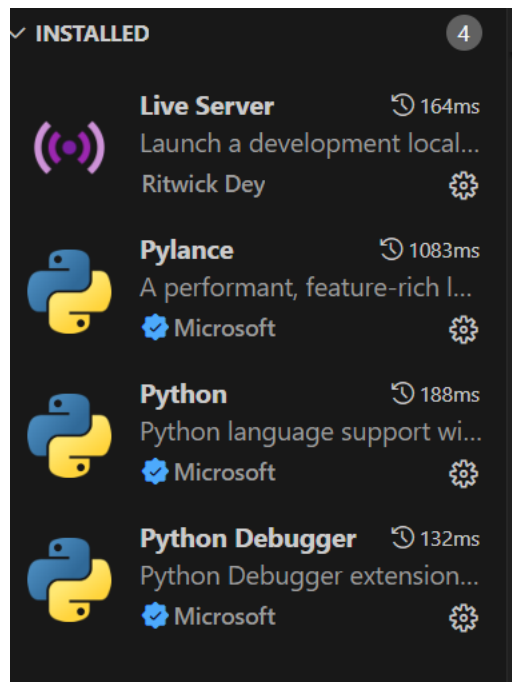


Рисунок 3.2. Встановлення "Add Python to PATH"

2. Встановлення Visual Studio Code

- 1) Завантажте та встановіть VS Code з [офіційного сайту VS Code](#).
- 2) Під час інсталяції можна вибрати опцію інтеграції у контекстне меню Windows (для зручності).

3. Налаштування VS Code для Python

Встановлення розширень. У VS Code, перейдіть до розділу Extensions (Ctrl + Shift + X) та встановіть:

- Python (офіційне розширення від Microsoft)
- PyQt Integration (для роботи з PyQt)
- Code Runner (для запуску коду в редакторі)
- Pylance (покращена підтримка автодоповнення)

4. Створення віртуального середовища (Virtual Environment)

- 1) Відкрийте термінал у VS Code (Ctrl + `).

- 2) Перейдіть у теку свого проєкту.
- 3) Створіть віртуальне середовище:

В терміналі VS Code потрібно прописати

```
python -m venv venv
```

- 4) Активуйте його:

Windows:

Після активації віртуального середовища у терміналі з'явиться **(venv)**.
(Рисунок 3.3.)

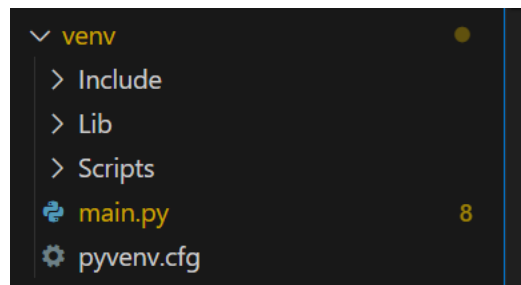


Рисунок 3.3. Результат пропису в терміналі команди

5. Встановлення необхідних бібліотек

У терміналі VS Code введіть команди для встановлення бібліотек:

```
pip install PyQt5 PySide6 requests
```

Додатково (Рисунок 3.4.):

- 1) Для веб-браузера у PySide6:

В терміналі VS Code потрібно прописати:

```
pip install PySide6-Addons PySide6-Essentials
```

2) Для налаштування форматування коду:

В терміналі VS Code потрібно прописати:

```
pip install black pylint
```

```
Installing collected packages: PyQt5-Qt5, urllib3, shiboken6, PyQt5-sip, idna,
Successfully installed PyQt5-5.15.11 PyQt5-Qt5-5.15.2 PyQt5-sip-12.16.1 PySide
requests-2.32.3 shiboken6-6.8.1 urllib3-2.3.0

[notice] A new release of pip is available: 24.0 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\Users\Acer Nitro 5\Desktop>
```

Рисунок 3.4. Налаштування

6. Налаштування інтерпретатора Python у VS Code

1. Натисніть Ctrl + Shift + P → знайдіть "Python: Select Interpreter".
2. Виберіть інтерпретатор віртуального середовища (**venv**) вашого проєкту.

7. Тестування PyQt5/PySide6

Створіть файл main.py і додайте тестовий код:

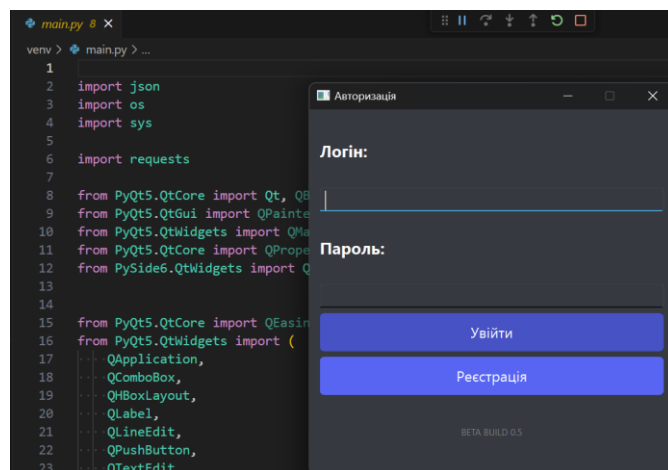


Рисунок 3.5. Авторизація.

Запустити код у VS Code: Натиснути F5 або "Run Python File" у верхній панелі.

8. Поради для роботи в VS Code

- 1) Форматування коду: Використовуйте black для автоматичного форматування.
- 2) Літинг: Увімкніть pylint у налаштуваннях (Settings → Python Linting).
- 3) Запуск скриптів: Використовуйте Run або Code Runner.
- 4) Налаштування терміналу: Виберіть "Python" Terminal для запуску у віртуальному середовищі.

Тепер середовище Python у VS Code повністю налаштоване для роботи з PyQt5, PySide6 та іншими бібліотеками.

3.3. Огляд власної розробки

1. Система авторизації була розроблена для захисту даних які використовуються під час роботи з програмою що є актуально в наш час.(Рисунок 3.6.)

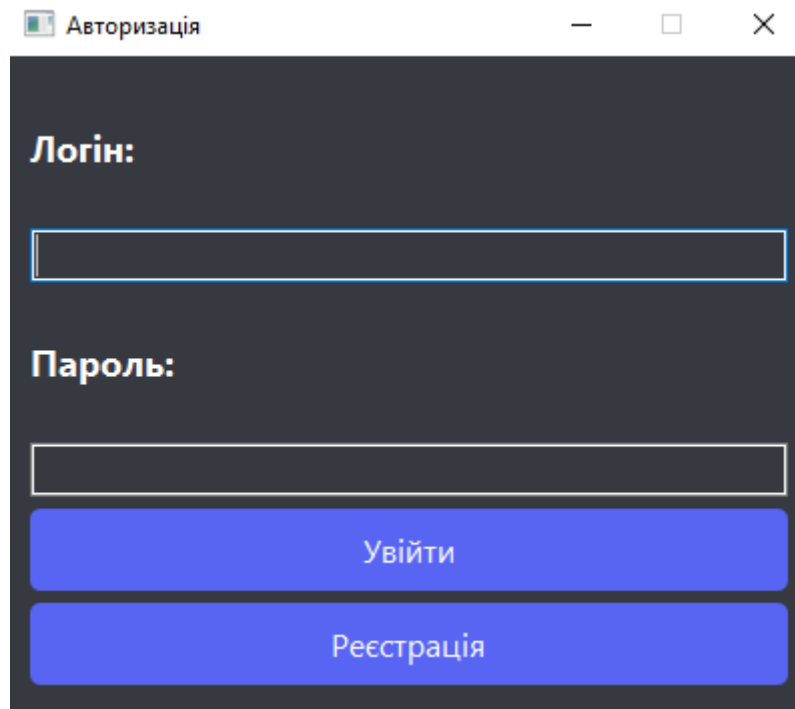


Рисунок 3.6. Система авторизації.

2. Інтерфейс програми – стандартний інтерфейс для простого сприйняття інформації.(Рисунок 3.7.)

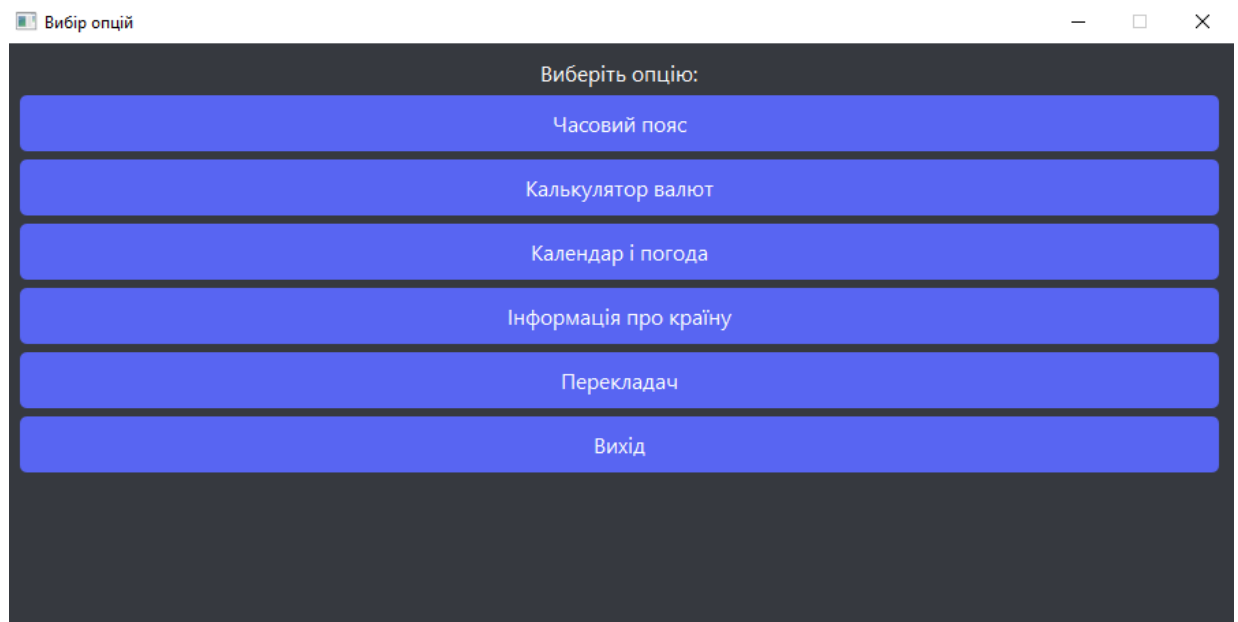


Рисунок 3.7. Вибір потрібної функції (опції)

3. Часовий пояс відображає інформацію яку бере через відкритий API часого поясу (Рисунок 3.8)

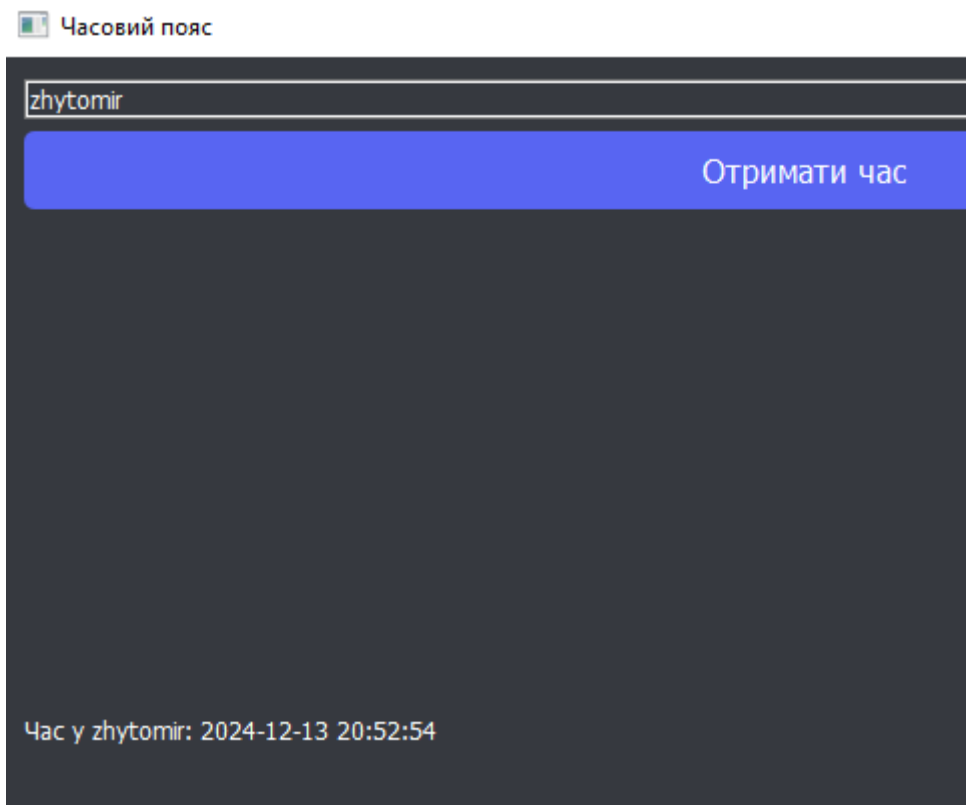


Рисунок 3.8. Часовий пояс

4. Калькулятор валют переводить певну валюту в іншу згідно НБУ (Рисунок 3.9.)

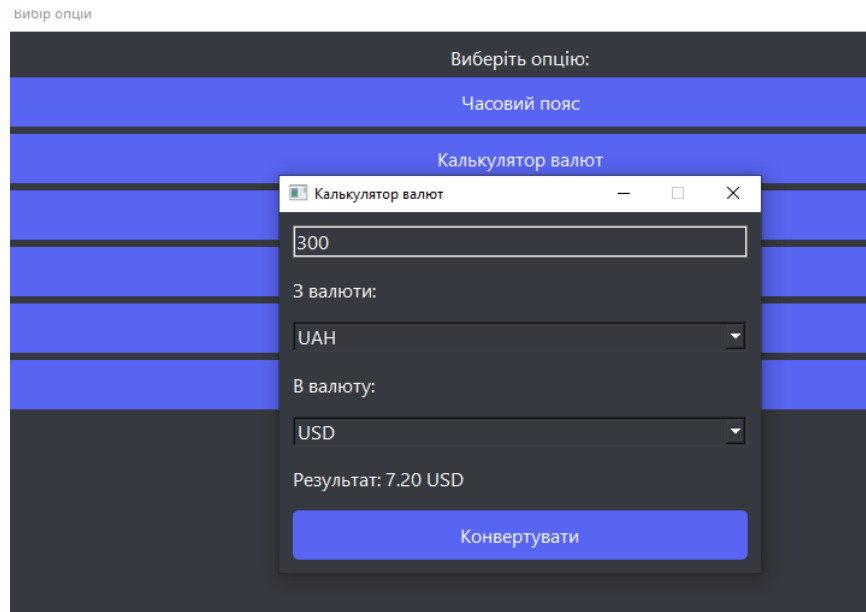


Рисунок 3.9. Калькулятор валют

5. Календар і погода працює в усьому світі, та через API передає інформацію про день та погоду яка відображається у нас в програмі (Рисунок 3.10.)

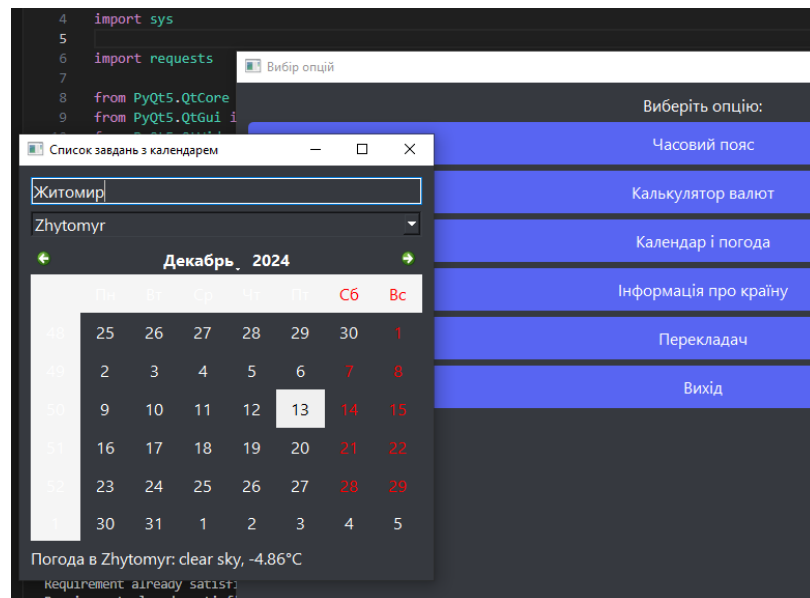


Рисунок 3.10. Календар та погода

6. Інформація про країну надає інформацію про столицю, населення, прапор та валюту (Рисунок 3.11.)

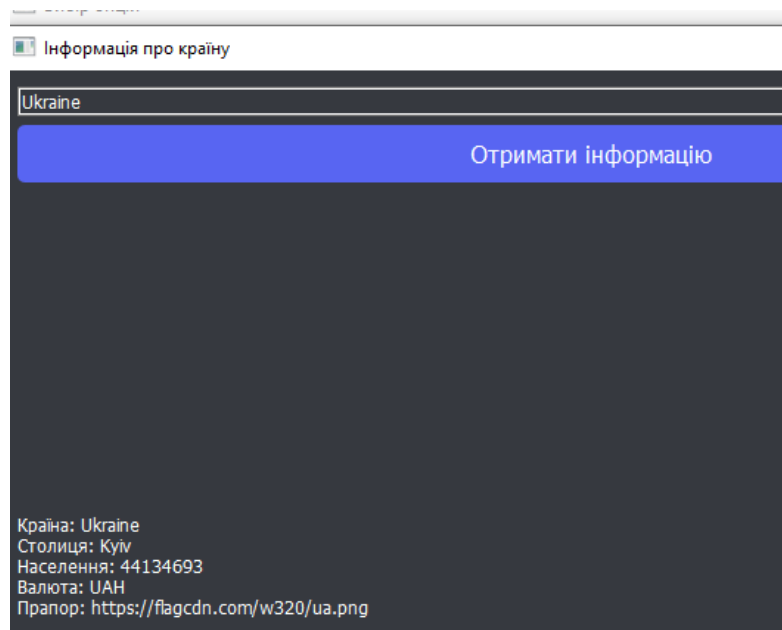


Рисунок 3.11. Інформація про країну

7. Перекладач перекладає за допомогою API з тих мов що ми обрали

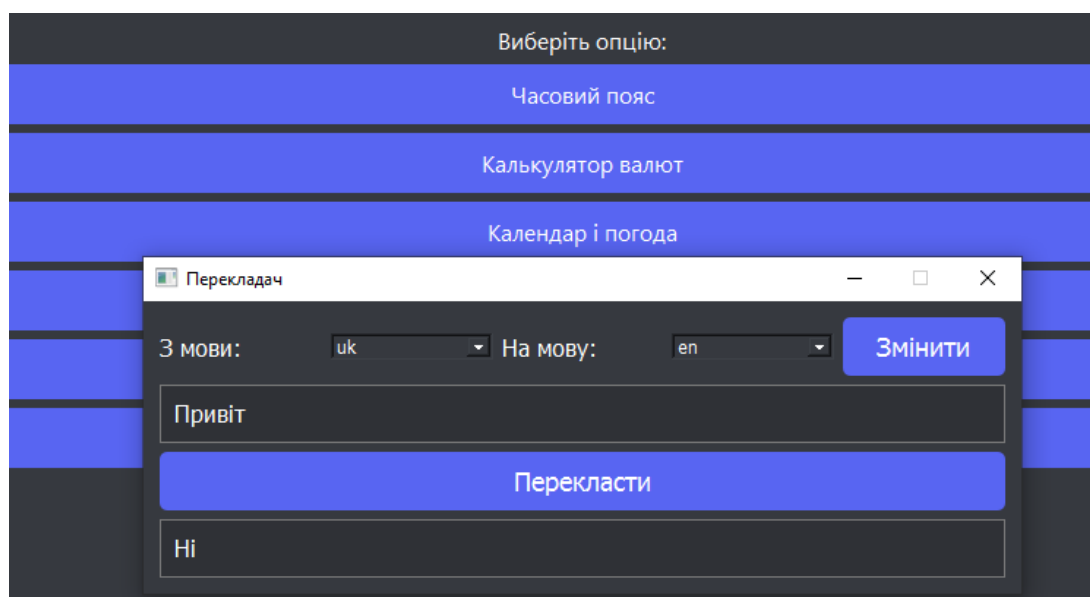


Рисунок 3.12. Перекладач

Висновок до розділу 3

У результаті реалізації проєкту було створено функціональну програму, яка об'єднує кілька важливих можливостей та інструментів для користувача. Використання **Visual Studio Code** як середовища розробки забезпечило ефективність роботи завдяки його гнучкості, швидкодії та потужному функціоналу.

Основні досягнення даного проєкту включають:

1. **Розробка системи авторизації** – реалізовано механізм захисту даних користувача, що є актуальним у сучасних умовах безпеки.
2. **Інтуїтивно зрозумілий інтерфейс** – забезпечено просте та доступне сприйняття інформації для користувача.
3. **Інтеграція API для часових поясів** – надання точної інформації про час у різних регіонах світу.
4. **Калькулятор валют** – створено можливість конвертації валют на основі актуальних даних НБУ.
5. **Календар та погода** – забезпечено отримання глобальної інформації про дати та погодні умови через відкриті API.
6. **Інформація про країну** – надано детальну інформацію про столицю, населення, прапор та валюту обраної країни.
7. **Перекладач мов** – реалізовано функціонал перекладу з обраних мов за допомогою API.

Проєкт демонструє ефективну інтеграцію **сучасних технологій** та **зовнішніх API**, що дозволило створити багатофункціональний додаток. Вибір **Python** як основної мови програмування та **VS Code** як середовища розробки забезпечив гнучкість та легкість у реалізації поставлених завдань.

ВИСНОВОК

У ході даної кваліфікаційної роботи було досліджено теоретичні основи та практичні аспекти використання API, веб-сервісів, їх тестування та впровадження у сучасних програмних системах. Розглянуто ключові технології, такі як SOAP, REST, WSDL, а також їхні особливості й застосування для інтеграції та взаємодії між програмними компонентами.

Основні результати роботи:

Визначено методи обміну даними через API та роль веб-сервісів у сучасних інформаційних системах. Детально вивчено SOAP та WSDL, які забезпечують стандартизовану взаємодію між клієнтом і сервером, а також можливість абстрагування інтерфейсів від їхньої реалізації.

Проаналізовано методології тестування веб-сервісів, включаючи функціональне, синтаксичне та інтеграційне тестування. Автоматизовані інструменти, такі як SoapUI і TestComplete, дозволяють підвищити ефективність виявлення помилок і забезпечують надійність роботи API у розподілених системах.

На базі Visual Studio Code було розроблено програму з функціональністю, що включає систему авторизації, інтеграцію з веб-сервісами для отримання даних про час, валюту, погоду та інформацію про країни. Використання Python як мови програмування забезпечило швидкість розробки, а API дозволило ефективно інтегрувати сторонні сервіси.

Таким чином, дана робота демонструє ефективне застосування теоретичних знань для створення практичних рішень. Впровадження API, тестування веб-сервісів та їх інтеграція дозволяють створювати гнучкі, надійні та багаторазово використовувані програмні системи, які відповідають сучасним вимогам якості та продуктивності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Eric Matthes. "Python Crash Course", 3rd Edition. No Starch Press, 2023.
2. Al Sweigart. "Automate the Boring Stuff with Python", 2nd Edition. No Starch Press, 2020.
3. Mark Summerfield. "Programming in Python 3", Addison-Wesley, 2010.
4. Python Software Foundation. Python Documentation. [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.python.org/>
5. Microsoft. Visual Studio Code User Documentation. [Електронний ресурс] – Режим доступу до ресурсу: <https://code.visualstudio.com/docs>
6. PyPI: Python Package Index – репозиторій для бібліотек Python. [Електронний ресурс] – Режим доступу до ресурсу: <https://pypi.org/>
7. S. Hu та T. Dai, "Online Map Application Development Using Google Maps API, SQL Database, and ASP.NET", International Journal of Information and Communication Technology Research, с. 102–103, 2013.
8. N. Chatterjee та S. Chakraborty, "Real-time Communication Application Based on Android Using Google Firebase", International Journal of Advance Research in Computer Science and Management Studies, с. 74–76, 2018.
9. C. Khawas, "Application of Firebase in Android App Development-A Study", International Journal of Computer Applications, June, с. 49–52, 2018.
10. М. Г. Завадський, "Основи програмування мовою Python", Київ: Видавництво, с. 120–125, 2020.
11. В. Бондар, "Програмування мовою Python. Створення застосунків", Львів: Видавничий дім, с. 95–102, 2021.
12. О. Чорненький та С. Бойко, "Python для початківців та професіоналів", Київ: Освіта, с. 150–157, 2019.
13. Ю. В. Погребний, "Інформаційні технології та програмування мовою Python", Київ: Академія, с. 78–85, 2022.

14. В. Пилипчук, "Проектування застосунків на Python. Основи та приклади", Івано-Франківськ: Техносфера, с. 110–115, 2020.
15. OpenWeather API та інтеграція в Python. [Електронний ресурс] – Режим доступу до ресурсу: <https://dou.ua/>
16. Національний банк України (НБУ). "Робота з API НБУ для отримання курсів валют" [Електронний ресурс] – Режим доступу до ресурсу: <https://bank.gov.ua/>

ДОДАТКИ

Лістинг програми:

```
import json
import os
import sys

import requests

from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QApplication, QVBoxLayout, QPushButton, QWidget

from PyQt5.QtWidgets import (
    QApplication,
    QComboBox,
    QHBoxLayout,
    QLabel,
    QLineEdit,
    QPushButton,
    QTextEdit,
    QVBoxLayout,
    QWidget,
)

from PySide6.QtCore import Qt
from PySide6.QtWidgets import (
    QApplication,
    QCalendarWidget,
    QComboBox,
    QHBoxLayout,
    QLabel,
    QLineEdit,
    QMessageBox,
    QPushButton,
    QTextEdit,
    QVBoxLayout,
    QWidget,
```

)

```
# Глобальні змінні
image_generator_window = None
country_info_window = None
selection_window = None
global_todo_window = None
web_view = None
auth_window = None
settings_window = None
weather_window = None
calendar_window = None
lists_window = None
schedule_window = None # Вікно для розкладу зустрічей

credentials_file = "credentials.txt"
notes_file = "notes.json"

window_size = (900, 600)
font_size = 16
theme_colors = {
    "background": "#36393F",
    "text": "#FFFFFF",
    "button_bg": "#5865F2",
    "button_hover": "#4752C4",
    "input_bg": "#40444B",
    "border": "#5865F2",
}

def apply_settings(window):
    window.setStyleSheet(f"""
    QWidget {{
        background-color: {theme_colors["background"]};
        color: {theme_colors["text"]};
        font-size: {font_size}px;
```

```

}}
QPushButton {{
    background-color: {theme_colors["button_bg"]};
    color: white;
    padding: 10px;
    border-radius: 5px;
}}
QPushButton:hover {{
    background-color: {theme_colors["button_hover"]};
}}
""")

```

Функція для перевірки облікових даних

```

def check_credentials():
    username = entry_username.text()
    password = entry_password.text()

    if os.path.exists(credentials_file):
        with open(credentials_file, "r") as f:
            saved_credentials = json.load(f)
            if username in saved_credentials and saved_credentials[username] == password:
                auth_window.close()
                open_selection_window()
                return

    if username == "123" and password == "123":
        auth_window.close()
        open_selection_window()
    else:
        QMessageBox.warning(auth_window, "Помилка", "Невірний логін або пароль")

def open_schedule_window():
    global schedule_window
    if schedule_window is None:
        schedule_window = QWidget()
        schedule_window.setWindowTitle("Часовий пояс")
        schedule_window.setGeometry(100, 100, 800, 600) # Збільшений розмір вікна

```

```

# Основний макет
layout = QVBoxLayout()

def get_time_by_city():
    city = city_input.text()
    api_key = " RYAZGVU2L2IQ" # Публічний ключ для TimeZoneDB
    url = f"http://api.timezonedb.com/v2.1/get-time-
zone?key={api_key}&format=json&by=zone&zone={city}"
    response = requests.get(url)

    if response.status_code == 200:
        data = response.json()
        if data["status"] == "OK":
            time_info = f"Час у {city}: {data['formatted']}"
        else:
            time_info = f"Помилка: {data['message']}"
    else:
        time_info = f"Не вдалося отримати дані про час. Код помилки:
{response.status_code}"

    time_label.setText(time_info)

# Поле для введення назви міста
city_input = QLineEdit()
city_input.setPlaceholderText("Введіть часовий пояс (наприклад, Europe/Kyiv)")
layout.addWidget(city_input)

# Кнопка для отримання часу
btn_get_time = QPushButton("Отримати час")
btn_get_time.clicked.connect(get_time_by_city)
layout.addWidget(btn_get_time)

# Мітка для відображення часу
time_label = QLabel("Час:")
layout.addWidget(time_label)

```

```

# Застосування стилів
set_common_style(schedule_window)

schedule_window.setLayout(layout)
schedule_window.show()

def set_common_style(widget):
    widget.setStyleSheet("""
        QWidget {
            background-color: #36393F;
            color: #FFFFFF;
            font-family: 'Whitney', sans-serif;
        }
        QPushButton {
            background-color: #5865F2;
            border-radius: 5px;
            color: white;
            padding: 10px;
            font-size: 16px;
        }
        QPushButton:hover {
            background-color: #4752C4;
        }
        QListWidget {
            background-color: #40444B;
            color: white;
        }
        QTextEdit {
            background-color: #40444B;
            color: white;
            border: 1px solid #5865F2;
        }
    """)

# Функція для відкриття вікна авторизації
def open_auth_window():
    global auth_window, entry_username, entry_password

```

```

auth_window = QWidget()
auth_window.setWindowTitle("Авторизація")
auth_window.setFixedSize(400, 400)
layout = QVBoxLayout()

label_username = QLabel("Логін:")
label_username.setStyleSheet("font-size: 18px; font-weight: bold;")
layout.addWidget(label_username)
entry_username = QLineEdit()
layout.addWidget(entry_username)

label_password = QLabel("Пароль:")
layout.addWidget(label_password)
label_password.setStyleSheet("font-size: 18px; font-weight: bold;")
entry_password = QLineEdit()
entry_password.setEchoMode(QLineEdit.Password)
layout.addWidget(entry_password)

btn_login = QPushButton("Увійти")

btn_login.clicked.connect(check_credentials)
layout.addWidget(btn_login)

btn_register = QPushButton("Реєстрація")
btn_register.clicked.connect(open_registration_window)
layout.addWidget(btn_register)
label_version = QLabel("BETA BUILD 0.5")
label_version.setAlignment(Qt.AlignCenter)
label_version.setStyleSheet("font-size: 10px; font-weight: normal; color: gray;")
layout.addWidget(label_version)

auth_window.setLayout(layout)

apply_settings(auth_window) # Застосовуємо стиль до вікна авторизації
auth_window.show()

```

```

def open_registration_window():
    global registration_window
    registration_window = QWidget()
    registration_window.setWindowTitle("Рєєстрація")
    registration_window.resize(400, 400)
    layout = QVBoxLayout()

    label_username = QLabel("Логін:")
    label_username.setStyleSheet("font-size: 18px; font-weight: bold;")
    layout.addWidget(label_username)
    reg_username = QLineEdit()
    layout.addWidget(reg_username)

    label_password = QLabel("Пароль:")
    label_password.setStyleSheet("font-size: 18px; font-weight: bold;")
    layout.addWidget(label_password)
    reg_password = QLineEdit()
    reg_password.setEchoMode(QLineEdit.Password)
    layout.addWidget(reg_password)

    btn_register = QPushButton("Зареєструватись")
    btn_register.clicked.connect(lambda: register_user(reg_username.text(), reg_password.text()))
    layout.addWidget(btn_register)

    registration_window.setLayout(layout)
    registration_window.setStyleSheet("""
        QWidget {
            background-color: #ADD8E6; /* Світло-синій колір */
            color: #000000; /* Чорний текст */
        }
        QPushButton {
            background-color: #5DADE2;
            color: white;
            padding: 10px;
            border-radius: 5px;
        }
    """)

```

```

        QPushButton: hover {
            background-color: #2874A6;
        }
    """)
    apply_settings(registration_window)
    registration_window.show()

def register_user(username, password):
    if username and password:
        if os.path.exists(credentials_file):
            with open(credentials_file, "r") as f:
                saved_credentials = json.load(f)
        else:
            saved_credentials = {}

        if username in saved_credentials:
            QMessageBox.warning(registration_window, "Помилка", "Цей логін уже зайнятий!")
        else:
            saved_credentials[username] = password
            with open(credentials_file, "w") as f:
                json.dump(saved_credentials, f)
            QMessageBox.information(registration_window, "Успіх", "Реєстрація успішна!")
            registration_window.close()
        else:
            QMessageBox.warning(registration_window, "Помилка", "Введіть логін і пароль")

def open_selection_window():
    global selection_window
    selection_window = QWidget()
    selection_window.setWindowTitle("Вибір опцій")
    selection_window.setFixedSize(*window_size)

    apply_settings(selection_window)

    layout = QVBoxLayout()

    label_message = QLabel("Виберіть опцію:")
    label_message.setAlignment(Qt.AlignCenter)

```

```
label_message.setStyleSheet(f"font-size: {font_size}px;")
layout.addWidget(label_message)
```

```
btn_schedule = QPushButton("Часовий пояс")
btn_schedule.clicked.connect(open_schedule_window)
layout.addWidget(btn_schedule)
```

```
btn_currency_calculator = QPushButton("Калькулятор валют")
btn_currency_calculator.clicked.connect(open_currency_calculator)
layout.addWidget(btn_currency_calculator)
```

```
btn_todo = QPushButton("Календар і погода") # Кнопка для TODO списку
btn_todo.clicked.connect(open_todo_window) # Виклик функції open_todo_window
layout.addWidget(btn_todo)
btn_country_info = QPushButton("Інформація про країну")
btn_country_info.clicked.connect(open_country_info_window)
layout.addWidget(btn_country_info)
```

```
btn_translator = QPushButton("Перекладач")
btn_translator.clicked.connect(open_translator) # Виклик функції open_translator
layout.addWidget(btn_translator)
```

```
btn_logout = QPushButton("Вихід")
btn_logout.clicked.connect(logout)
layout.addWidget(btn_logout)
```

```
layout.setAlignment(Qt.AlignTop)
selection_window.setLayout(layout)
selection_window.show()
```

```
def open_country_info_window():
    global country_info_window
    if country_info_window is None:
```

```

country_info_window = QWidget()
country_info_window.setWindowTitle("Інформація про країну")
country_info_window.setFixedSize(800, 600)

layout = QVBoxLayout()

country_input = QLineEdit()
country_input.setPlaceholderText("Введіть назву країни англійською")
layout.addWidget(country_input)

btn_get_info = QPushButton("Отримати інформацію")
layout.addWidget(btn_get_info)

country_info_label = QLabel("Інформація: ")
layout.addWidget(country_info_label)

btn_get_info.clicked.connect(lambda: get_country_info(country_input.text(),
country_info_label))

set_common_style(country_info_window) # додайте стиль
country_info_window.setLayout(layout)
country_info_window.show()

def get_country_info(country_name, label):
    country_name = country_name.strip() # Видаляє зайві пробіли
    if not country_name:
        label.setText("Будь ласка, введіть назву країни.")
        return

    url = f"https://restcountries.com/v3.1/name/{country_name}?fullText=true"
    response = requests.get(url)
    if response.status_code == 200:
        country_info = response.json()[0]
        info = f"Країна: {country_info['name']['common']}\n"
        info += f"Столиця: {country_info['capital'][0] if 'capital' in country_info else 'Немає"
        info += f"даних'}\n"
        info += f"Населення: {country_info['population']}\n"

```

```

        info += f"Валюта: {'', '.join([currency for currency in country_info['currencies'].keys()])}\n"
        info += f"Прапор: {country_info['flags']['png']}"
        label.setText(info)
    else:
        label.setText("Не вдалося отримати інформацію про країну. Перевірте правильність введення.")

def open_todo_window():
    global global_todo_window
    global_todo_window = TodoApp()
    apply_settings(global_todo_window)
    global_todo_window.show()

class TodoApp(QWidget):
    def __init__(self):
        super().__init__()
        self.tasks = {}
        self.cities = []
        self.initUI()

    def initUI(self):
        layout = QVBoxLayout()

        # Поле для вводу назви міста
        self.city_input = QLineEdit(self)
        self.city_input.setPlaceholderText("Введіть назву міста")
        self.city_input.textChanged.connect(self.update_city_list)
        layout.addWidget(self.city_input)

        # Комбо-бокс для вибору міста
        self.city_combo = QComboBox(self)
        layout.addWidget(self.city_combo)

        # Календар
        self.calendar = QCalendarWidget(self)
        layout.addWidget(self.calendar)

        # Погода

```

```

self.weather_label = QLabel("Порода: ", self)
layout.addWidget(self.weather_label)

self.setLayout(layout)
self.setWindowTitle("Список завдань з календарем")
self.setGeometry(300, 300, 400, 400)

def update_city_list(self):
    query = self.city_input.text()
    if query:
        self.get_city_suggestions(query)

def get_city_suggestions(self, query):
    api_key = "bd5e378503939ddaee76f12ad7a97608"
    url =
f"http://api.openweathermap.org/data/2.5/find?q={query}&appid={api_key}&units=metric&cnt=
5"
    response = requests.get(url)

    if response.status_code == 200:
        data = response.json()
        self.cities = [city["name"] for city in data["list"]]
        self.city_combo.clear()
        self.city_combo.addItems(self.cities)
        self.city_combo.currentIndexChanged.connect(self.get_weather)
    else:
        print("Не вдалося отримати дані про міста.")

def get_weather(self):
    city = self.city_combo.currentText() # Отримати вибране місто
    api_key = "bd5e378503939ddaee76f12ad7a97608" # Ваш API ключ
    url = (
        f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=me
tric"
    )

    response = requests.get(url)

```

```

# Вивід статусу та тексту відповіді для налагодження
if response.status_code != 200:
    print(f"Error {response.status_code}: {response.text}")

if response.status_code == 200:
    data = response.json()
    weather_description = data["weather"][0]["description"]
    temperature = data["main"]["temp"]
    weather_info = f"Погода в {city}: {weather_description}, {temperature}°C"
else:
    weather_info = "Не вдалося отримати дані про погоду."

self.weather_label.setText(weather_info)
class Translator(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Перекладач")
        self.setFixedSize(600, 200)

    # Основний макет
    layout = QVBoxLayout()

    # Горизонтальний макет для мов
    language_layout = QHBoxLayout()

    # Вибір мови
    self.source_language_box = QComboBox()
    self.target_language_box = QComboBox()

    # Додайте мови до випадаючих списків
    languages = ["uk", "en", "es"] # Українська, Англійська, Іспанська
    self.source_language_box.addItem(languages)
    self.target_language_box.addItem(languages)

    language_layout.addWidget(QLabel("3 мови:"))
    language_layout.addWidget(self.source_language_box)

```

```

language_layout.addWidget(QLabel("На мову:"))
language_layout.addWidget(self.target_language_box)

# Кнопка для зміни мов місцями
swap_button = QPushButton("Змінити")
swap_button.clicked.connect(self.swap_languages)
language_layout.addWidget(swap_button)

layout.addLayout(language_layout)

# Поле для введення тексту
self.input_text = QTextEdit()
layout.addWidget(self.input_text)

# Кнопка для перекладу
translate_button = QPushButton("Перекласти")
translate_button.clicked.connect(self.translate_text)
layout.addWidget(translate_button)

# Поле для відображення результату
self.result_text = QTextEdit()
self.result_text.setReadOnly(True)
layout.addWidget(self.result_text)

self.setLayout(layout)

# Налаштування стилю
self.setStyleSheet("""
    QWidget {
        background-color: #36393F;
        color: #FFFFFF;
        font-family: 'Whitney', sans-serif;
    }
    QPushButton {
        background-color: #5865F2;
        border-radius: 5px;
        color: white;
    }

```

```

        font-size: 18px;
        padding: 10px;
    }
    QPushButton:hover {
        background-color: #4752C4;
    }
    QTextEdit {
        background-color: #2F3136;
        color: white;
        font-size: 16px;
        padding: 5px;
    }
    QLabel {
        font-size: 16px;
    }
    """)

```

```

def translate_text(self):
    source_language = self.source_language_box.currentText()
    target_language = self.target_language_box.currentText()
    text = self.input_text.toPlainText()

    if text:
        # Для демонстрації використовується безкоштовний API
        url =
f"https://api.mymemory.translated.net/get?q={text}&langpair={source_language}|{target_language}"
        ge}"

        try:
            response = requests.get(url)
            translated_text = response.json().get("responseData", {}).get("translatedText", "")
            self.result_text.setText(translated_text)
        except Exception:
            self.result_text.setText("Помилка перекладу.")

def swap_languages(self):
    current_source = self.source_language_box.currentText()
    current_target = self.target_language_box.currentText()

```

```

        self.source_language_box.setCurrentText(current_target)
        self.target_language_box.setCurrentText(current_source)

# Запуск програми
def open_translator():
    global translator_window
    translator_window = Translator() # Використовуйте клас Translator
    translator_window.show() # Показати вікно перекладача

class CurrencyCalculator(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Калькулятор валют")
        self.setFixedSize(400, 300)

        self.layout = QVBoxLayout()
        self.setLayout(self.layout)

        self.amount_input = QLineEdit()
        self.amount_input.setPlaceholderText("Введіть суму")
        self.layout.addWidget(self.amount_input)

        self.from_currency = QComboBox()
        self.to_currency = QComboBox()
        self.layout.addWidget(QLabel("З валюти:"))
        self.layout.addWidget(self.from_currency)
        self.layout.addWidget(QLabel("В валюту:"))
        self.layout.addWidget(self.to_currency)

        self.result_label = QLabel("Результат: ")
        self.layout.addWidget(self.result_label)

        self.convert_button = QPushButton("Конвертувати")
        self.convert_button.clicked.connect(self.convert_currency)
        self.layout.addWidget(self.convert_button)

        self.populate_currency_options()

```

```

def populate_currency_options(self):
    currencies = [
        "USD",
        "EUR",
        "GBP",
        "UAH",
        "JPY",
        "GBP",
        "CHF",
        "KZT",
    ] # Додайте більше валют за потребою
    self.from_currency.addItem(currencies)
    self.to_currency.addItem(currencies)

def convert_currency(self):
    amount = float(self.amount_input.text())
    from_currency = self.from_currency.currentText()
    to_currency = self.to_currency.currentText()

    if from_currency == to_currency:
        self.result_label.setText(f"Результат: {amount} {to_currency}")
        return

    try:
        # Отримання курсу валют
        response = requests.get(f"https://api.exchangerate-api.com/v4/latest/{from_currency}")
        data = response.json()
        rate = data["rates"][to_currency]
        converted_amount = amount * rate
        self.result_label.setText(f"Результат: {converted_amount:.2f} {to_currency}")
    except Exception as e:
        QMessageBox.warning(self, "Помилка", f"Не вдалося отримати курс валют: {e}")

def open_currency_calculator():
    global currency_calculator_window
    currency_calculator_window = CurrencyCalculator()
    apply_settings(currency_calculator_window) # Застосування теми до калькулятора валют

```

```
        currency_calculator_window.show()
def logout():
    selection_window.close()
    open_auth_window()
if __name__ == "__main__":
    app = QApplication(sys.argv)
    open_auth_window()
    sys.exit(app.exec())
```