УДК 004.415.2:004.434:004.78:656.02

С.В.Затолокіна
С.О.Жуков

РОЗРОБКА КОНСОЛЬНОГО ЗАСТОСУНКУ ІЗ ЗАСТОСУВАННЯМ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ НА ПРИКЛАДІ ПЛАНУВАЛЬНИКА ТУРИСТИЧНИХ ПОДОРОЖЕЙ

Вінницький національний технічний університет

Анотація

У роботі досліджено науково-практичні аспекти проєктування та програмної реалізації інформаційних систем для автоматизації індивідуальної туристичної логістики. Запропоновано архітектурне рішення у вигляді консольного застосунку (CLI) на базі мови програмування Python, яке є відмовостійкою та автономною альтернативою існуючим хмарним сервісам. На основі об'єктно-орієнтованого моделювання (UML) розроблено ієрархічну структуру бізнес-сутностей із впровадженням поліморфної поведінки та механізмів інкапсуляції зі строгою валідацією стану. Особливу увагу приділено концепції управління транзакційними процесами бронювання через кастомні менеджери контексту та забезпеченню персистентності об'єктних графів за допомогою технологій багатоформатної серіалізації (JSON, Pickle, CSV). Доведено ефективність застосування специфічних засобів мови Python для створення надійного, локально орієнтованого програмного забезпечення.

Ключові слова: об'єктно-орієнтоване програмування, Python, консольний застосунок, UML-моделювання, менеджер контексту, серіалізація, персистентність, відмовостійкість архітектури.

Abstract

The paper investigates the scientific and practical aspects of designing and implementing information systems for the automation of individual travel logistics. An architectural solution in the form of a command-line application (CLI) developed in Python is proposed as a reliable, fault-tolerant, and autonomous alternative to existing cloud-based services. Based on object-oriented modeling (UML), a hierarchical structure of business entities has been designed, incorporating polymorphic behavior and encapsulation mechanisms with strict state validation. Particular attention is paid to the management of booking transactions through custom context managers and to ensuring the persistence of object graphs using multi-format serialization technologies (JSON, Pickle, and CSV). The effectiveness of applying advanced Python language features for the development of reliable, locally oriented software systems has been demonstrated.

Keywords: object-oriented programming, Python, command-line application (CLI), UML modeling, context manager, serialization, persistence, architecture fault tolerance.

Вступ

У сучасних умовах обсяги інформації у сфері туристичних послуг постійно зростають, що вимагає автоматизації логістики, бронювання та комплексного обліку індивідуальних турів. Процес планування ускладнюється великою кількістю взаємопов'язаних параметрів: логістикою транспортних засобів, вибором проживання та розкладом екскурсій. Існуючі вебплатформи (Booking.com, TripIt, Expedia) мають функціональні обмеження для наскрізного планування через вузьку спеціалізацію, жорстку залежність від інтернет-з'єднання або надто комерціалізований та важкий графічний інтерфейс.

Створення локального консольного застосунку (CLI) на основі парадигми об'єктно-орієнтованого програмування (ООП) дозволяє абстрагуватися від побудови графічного інтерфейсу та сфокусуватися на проєктуванні гнучкої, відмовостійкої архітектури бізнес-логіки.

Результати дослідження

На етапі проєктування системи було проведено декомпозицію предметної області туристичного планування. Функціональні можливості розроблюваного програмного забезпечення охоплюють додавання нових турів, перегляд наявної інформації, видалення елементів із системи, створення та обробку бронювань, а також пошук, сортування, фільтрацію, розрахунок вартості послуг, збереження й завантаження даних та формування аналітичної звітності. З погляду користувача основними сценаріями використання системи є додавання нового туру, пошук і перегляд етапів, оформлення бронювання, редагування або видалення інформації та формування фінальних звітів. Усі ці процеси формалізовано за допомогою Use Case діаграми додатку, яка дозволила визначити межі системи та логіку взаємодії користувача з планувальником.

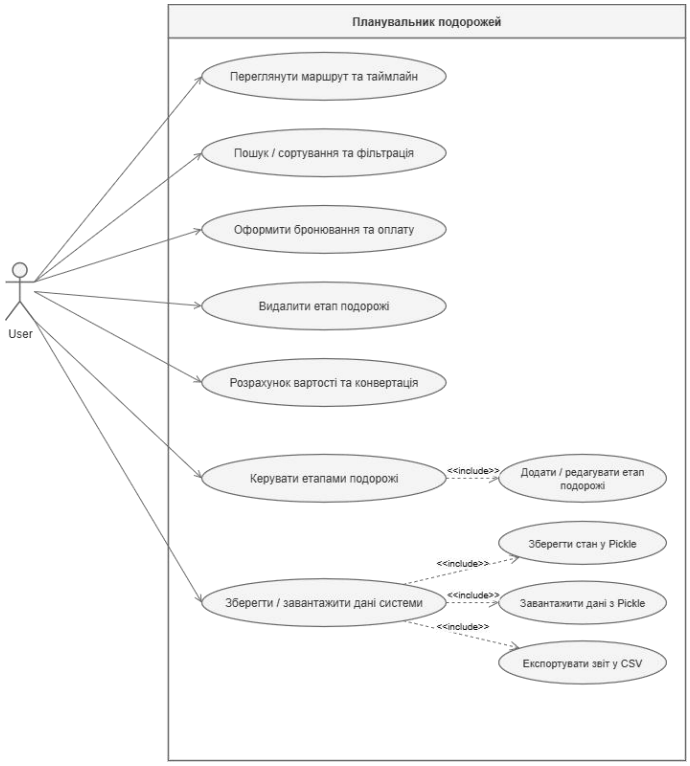


Рисунок 1. Use Case діаграма додатку.

Статичну структуру програмної моделі та зв'язки між компонентами відображено на діаграмі класів UML, яка демонструє організацію системи в термінах об'єктно-орієнтованого програмування. Архітектура застосунку базується на принципі композиції, де головним керуючим елементом виступає клас-менеджер Itinerary. Він містить у собі динамічну колекцію steps у вигляді списку об'єктів інших класів, повністю керує їхнім життєвим циклом і не може існувати у відриві від поточної структури маршруту.

Окрім композиції, на спроєктованій діаграмі впроваджено відношення наслідування між базовим класом TravelStep та його трьома нащадками — Flight, HotelBooking та Excursion. Також реалізовано спрямовану асоціацію, яка поєднує клас Itinerary із функтором CurrencyConverter, що використовується для динамічного перерахунку базової вартості квитків та готелів з іноземної валюти у гривневий еквівалент.

Відношення залежності зв'язує клас логістики Flight із перелічуваним типом TransportType, оскільки він залежить від фіксованих констант для суворої валідації доданого транспорту. Тимчасовий зв'язок обходу та створення пов'язує клас Itinerary з кастомним об'єктом ChronologicalIterator, який бере посилання на кроки подорожі для їх послідовного перегляду.

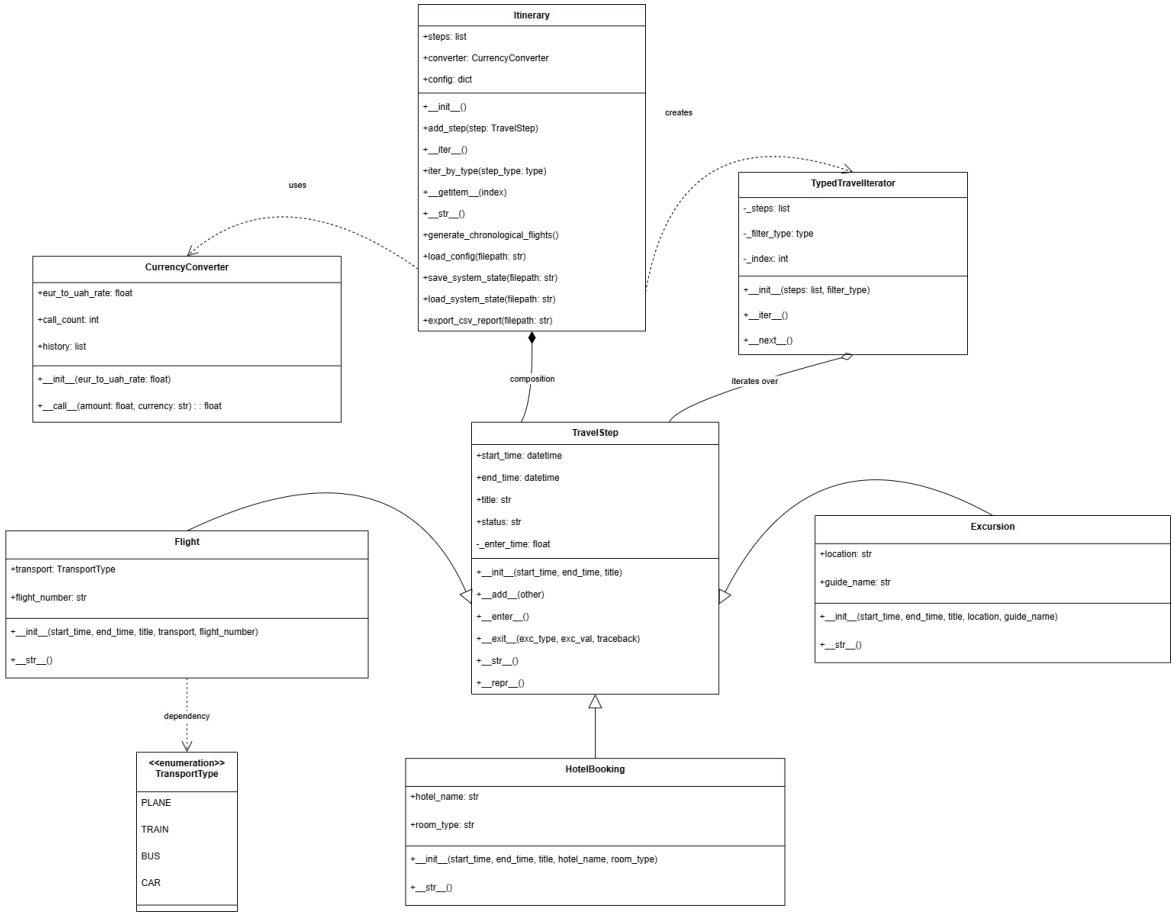


Рисунок 2 – UML діаграма класів системи

Програмна реалізація бізнес-сутностей виконана мовою Python з урахуванням висунутих технічних вимог. Для моделювання статичних даних, що не містять складної логіки (дані мандрівника Traveler та конфігурація системи AppConfig), застосовано декоратор @dataclass, що дозволило суттєво оптимізувати структуру коду.

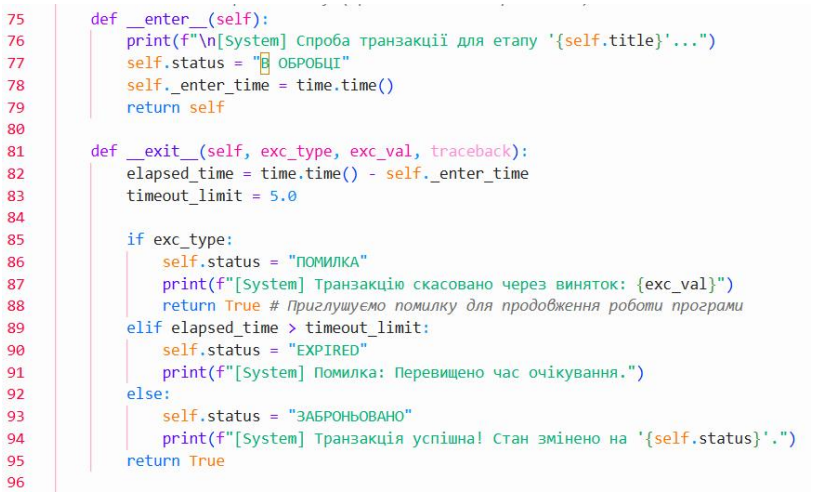
Механізм інкапсуляції та захисту стану реалізовано в класичному базовому класі TravelStep, де внутрішній фінансовий атрибут ціни _price захищений від прямого доступу. Взаємодія з ним відбувається виключно через декоратори @property та @price.setter, де реалізовано обов'язкову бізнес-валідацію: спроба встановити від'ємне значення викликає програмний виняток ValueError.

```
43  class TravelStep:
44      def __init__(self, start_time: datetime, end_time: datetime, title: str, price: float):
45          if end_time <= start_time:
46              raise ValueError("Час завершення етапу не може передувати часу початку.")
47          self.start_time = start_time
48          self.end_time = end_time
49          self.title = title
50          self.status = "ОЧІКУЄ"
51
52          # Інкапсуляція: захищений атрибут
53          self._price = 0.0
54          self.price = price # Виклик сетера для валідації
55          self._enter_time = None
56
57      @property
58      def price(self):
59          return self._price
60
61      @price.setter
62      def price(self, value):
63          if value < 0:
64              raise ValueError("Помилка валідації: Вартість не може бути від'ємною!")
65          self._price = value
66
```

Рисунок 3. Приклад з коду реалізації інкапсуляції та валідації в класі TravelStep

Ієрархія класів побудована на базі наслідування від TravelStep. Класи-нащадки Flight, HotelBooking та Excursion розширюють батьківський клас унікальними атрибутами предметної області. Поліморфна поведінка забезпечується через перевизначення методів get_icon() та get_details(). Це дозволяє класу-менеджеру взаємодіяти з різнорідними об'єктами колекції уніфіковано: під час рендерингу таймлайну кожен об'єкт самостійно повертає свій унікальний графічний маркер та текстові деталі.

Важливою архітектурною особливістю системи є транзакційне керування процесом оплати, реалізоване через власний менеджер контексту за допомогою магічних методів __enter__ та __exit__ у класі TravelStep. При вході в контекстний блок with етап переходить у стан «В ОБРОБЦІ» та фіксується час початку операції. Метод __exit__ здійснює контроль за виконанням: якщо імітація з'єднання перевищує ліміт часу, етапу присвоюється статус «EXPIRED», а у разі виникнення будь-якої помилки менеджер перехоплює виняток, змінює стан на «ПОМИЛКА» та приглушує його, повертаючи True. Це відображає концепцію Fault Tolerance та запобігає аварійному завершенню роботи всього CLI-інтерфейсу.



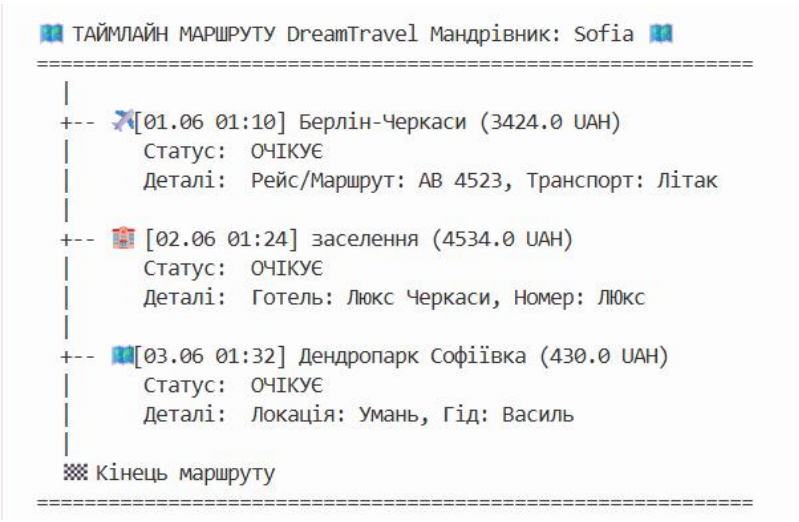
```
75      def __enter__(self):
76          print(f"\n[System] Спроба транзакції для етапу '{self.title}'...")
77          self.status = "В ОБРОБЦІ"
78          self._enter_time = time.time()
79          return self
80
81      def __exit__(self, exc_type, exc_val, traceback):
82          elapsed_time = time.time() - self._enter_time
83          timeout_limit = 5.0
84
85          if exc_type:
86              self.status = "ПОМИЛКА"
87              print(f"[System] Транзакцію скасовано через виняток: {exc_val}")
88              return True # Приглушуємо помилку для продовження роботи програми
89          elif elapsed_time > timeout_limit:
90              self.status = "EXPIRED"
91              print(f"[System] Помилка: Перевищено час очікування.")
92          else:
93              self.status = "ЗАБРОНЬОВАНО"
94              print(f"[System] Транзакція успішна! Стан змінено на '{self.status}'.")
95          return True
96
```

Рисунок 4. Контекстний менеджер транзакцій бронювання.

Для забезпечення правильної топології відображення маршруту розроблено кастомний ітератор ChronologicalIterator, який обходить кроки подорожі у хронологічному порядку.

Сортування подій працює завдяки перевизначенню магічного методу порівняння __lt__ за атрибутом часу початку події. Перевизначення магічного методу __str__ у класі Itinerary дозволяє автоматично малювати красиву графічну стрічку часу безпосередньо в консолі користувача. Додатково реалізовано магічний метод __getitem__ для швидкого доступу до відсортованих елементів за їхнім індексом.

Персистентність та стійкість даних між запусками програми забезпечується трирівневою системою. Збереження налаштувань застосунку (назва агенції, курс валют) реалізовано у форматі JSON через файл config.json. Консервація повного графа об'єктів системи та поточних статусів здійснюється за допомогою бінарної серіалізації модулем pickle у файл travel_data.pkl. Для формування аналітичної звітності розроблено функцію експорту даних у плоский табличний формат itinerary_report.csv із кодуванням utf-8-sig, що забезпечує коректне відображення кирилиці в Microsoft Excel. Користувацький CLI-інтерфейс реалізовано через нескінченний цикл while із текстовим меню на основі оператора match/case. Глобальне перехоплення винятків (try...except) захищає систему від випадкових помилок введення даних оператором, стабілізуючи роботу програми.



```
🗺️ ТАЙМЛАЙН МАРШРУТУ DreamTravel Мандрівник: Sofia 🗺️
============================================================
  |
  +-- ✈️[01.06 01:10] Берлін-Черкаси (3424.0 UAH)
  |       Статус:  ОЧІКУЄ
  |       Деталі:  Рейс/Маршрут: АВ 4523, Транспорт: Літак
  |
  +-- 🏨 [02.06 01:24] заселення (4534.0 UAH)
  |       Статус:  ОЧІКУЄ
  |       Деталі:  Готель: Люкс Черкаси, Номер: ЛЮкс
  |
  +-- 🗺️[03.06 01:32] Дендропарк Софіївка (430.0 UAH)
  |       Статус:  ОЧІКУЄ
  |       Деталі:  Локація: Умань, Гід: Василь
  |
  🏁 Кінець маршруту
============================================================
```

Рисунок 5. Приклад консольного виводу результату роботи.

Висновки

У результаті проведеного дослідження було успішно розроблено відмовостійкий консольний застосунок «Планувальник туристичних подорожей», який автоматизує логістичні процеси формування індивідуальних турів. Проведена робота підтвердила ефективність використання об'єктно-орієнтованого підходу під час створення програмних систем даного класу. Застосування UML-моделювання та діаграми варіантів використання дозволило сформувати чітку структуру програмного забезпечення з розмежуванням відповідальності між окремими компонентами системи, а використання композиції забезпечило надійне керування колекціями даних та взаємозв'язками між об'єктами.

Реалізація основних принципів об'єктно-орієнтованого програмування, зокрема інкапсуляції з механізмами валідації даних та поліморфного наслідування, сприяла підвищенню цілісності бізнес-даних і забезпечила можливість уніфікованої роботи з різними типами сутностей без ускладнення програмного коду. Важливим результатом стало впровадження власних контекстних менеджерів на основі методів __enter__ та __exit__, що дозволило реалізувати безпечне моделювання транзакційних процесів бронювання, забезпечити коректну обробку виняткових ситуацій та підвищити загальну стійкість програмного комплексу до помилок.

У ході роботи також було розроблено багаторівневу підсистему персистентності, яка поєднує використання форматів JSON, Pickle та CSV. Отримані результати демонструють, що створений програмний комплекс характеризується високою відмовостійкістю, швидкодією та гнучкістю архітектури, а його структура створює надійну основу для подальшого функціонального розширення та інтеграції в інформаційні системи туристичних агенцій.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Посилання на GitHub репозиторій з кодом застосунку
   URL: https://github.com/zatolokinasofia74/planyvalnik_podorojey
2. Соммервілл І. Програмна інженерія. Об'єктно-орієнтований подход / Іан Соммервілл. – 10-те видання. – К. : Вільямс, 2016. – 816 с.
3. Якобсон І. Об'єктно-орієнтоване програмне інженерство: процес, орієнтований на використання / Івар Якобсон. – К. : Вільямс, 2016. – 512 с.
4. Лавріненко О. В. Алгоритми та структури даних : навчальний посібник / О. В. Лавріненко. – К. : КПІ ім. Ігоря Сікорського, 2018. – 256 с.
5. Мокін Б. І. Навчальний посібник для опанування студентами способів розв'язання задач з функціонального аналізу мовою Python. Частина 1 / Б. І. Мокін, В. Б. Мокін, О. Б. Мокін. – Вінниця : ВНТУ, 2024. URL: https://press.vntu.edu.ua/
6. Інтелектуальний аналіз даних та машинне навчання. Частина 1. Базові методи та засоби аналізу даних : навчальний посібник / Вінницький національний технічний університет. – Вінниця : ВНТУ, 2023. URL: https://iq.vntu.edu.ua/
7. Python Language Reference and Standard Library Documentation [Електронний ресурс]. URL: https://docs.python.org/
8. Unified Modeling Language (UML) Specification [Електронний ресурс] / Object Management Group. URL: https://www.omg.org/uml/
9. Visual Studio Code – Code Editing. Redefined [Електронний ресурс]. URL: https://code.visualstudio.com/
10. Діаграма класів [Електронний ресурс] // Вікіпедія : вільна енциклопедія. URL: https://uk.wikipedia.org/wiki/Діаграма_класів
11. Алгоритм [Електронний ресурс] // Вікіпедія : вільна енциклопедія. URL: https://uk.wikipedia.org/wiki/Алгоритм

· ***Затолокіна Софія Вадимівна*** – студентка групи СА-25б, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м.Вінниця, e-mail: zatolokinasofia74@gmail.com

Жуков Сергій Олександрович — *к.т.н., доцент кафедри системного аналізу та інформаційних технологій, Вінницький національний технічний університет, e-mail: sazhukov@gmail.com*

Zatolokina Sofia V. – student of group SA-25b, Faculty of Intelligent Information Technologies and Automation, Vinnytsia National Technical University, Vinnytsia, e-mail: zatolokinasofia74@gmail.com

Zhukov Serhii O. — Ph.D., Assistant Professor of the Department of Systems Analysis and Information Technologies, Vinnytsia National Technical University, Vinnytsia, e-mail: sazhukov@gmail.com