НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри інформаційних систем і технологій

_____/Швиденко М.З.,доц.,к.е.н.
підпис ПІБ, вчене звання і ступінь

«__»_____2025р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

«Інформаційно-допоміжна система для підбору та замовлень продуктів»

Спеціальність 122 «Комп'ютерні науки»

Гарант освітньої програми

Д.е.н., професор _____ Руденський Р.А.

Керівник бакалаврської кваліфікаційної роботи

Ст. викл. _____ Золотуха Р.А.

Виконала _____ Карпенко К.М.

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Затверджую
Завідувач кафедри інформаційних
систем і технологій
_____/Швиденко М.З.,доц.,к.е.н.
підпис ПІБ, вчене звання і ступінь

«__»_____202_р

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студентці Карпенко Катерині Максимівні
спеціальності 122 «Комп'ютерні науки»
Тема бакалаврської кваліфікаційної роботи: **Інформаційно-допоміжна система для підбору та замовлень продуктів**
Затверджена наказом ректора НУБіП України від 26.02. 2025р №237 "С"
Термін подання завершеної роботи на кафедру _____
(рік,місяць,число)
Вихідні дані до бакалаврської кваліфікаційної роботи:
Перелік питань, які потрібно розробити:

1. Системний аналіз предметної області
2. Інформаційне забезпечення
3. Прикладне програмне забезпечення
4. Рекомендації щодо впровадження та експлуатації системи

Дата видачі завдання "26" лютого 2025 р.
Керівник бакалаврської кваліфікаційної роботи: ст. викл._____Золотуха Р.А.
Завдання прийняла до виконання _____ Карпенко К.М.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 5

ВСТУП 6

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 8

1.1 Постановка завдання 8

1.2 Огляд інформаційних джерел та існуючих рішень 9

1.3 Моделювання предметної області 12

1.4 Функціональні та нефункціональні вимоги 16

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ 18

2.1 Формат і структура файлу recipes.json 18

2.2 Excel-файл orders.xlsx як проміжне сховище даних 19

2.3 Структура БД Blue Prism (SQL Server) і робоча черга 20

2.4 Проєктування черги (поля, SLA, стани, пріоритети) 22

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ 25

3.1 Реалізація Telegram-бота 25

3.1.1 Вибір технологічного стеку (Python + aiogram) 25

3.1.2 Архітектура бота та модулі (config.py, MyOrderBot.py) 26

3.1.3 Пошук страв та формування замовлення (парсинг JSON) 34

3.1.4 Експорт замовлення у orders.xlsx та передача до BP-черги 37

3.2 Реалізація RPA-рівня в Blue Prism 39

3.2.1 Теоретичні основи Blue Prism (архітектура платформи, Process vs Object Studio, Queues) 39

3.2.2 Бізнес-об’єкт «Silpo Web Shop BO» (пошук, кошик) 52

3.2.3 Процес Create Grocery Order 54

3.2.4 Налаштування Control Room (ресурси, моніторинг) 56

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ 58

4.1 Тестування системи 58

4.1.1 План тестування (unit → integration) 58

4.1.2 Тестування Telegram-бота та показники покриття 59

4.1.3 Тестування процесу Blue Prism 62

4.2 Вимоги до апаратного та програмного забезпечення 67

4.3 Склад інсталяційного пакету 68

ВИСНОВОК 71

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 73

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1) API – Application Programming Interface
2) BO – Business Object
3) BP – Blue Prism
4) GUI – Graphical User Interface
5) HTML – HyperText Markup Language
6) JSON – JavaScript Object Notation
7) KPI – Key Performance Indicator
8) RPA – Robotic Process Automation
9) SLA – Service Level Agreement
10) SQL – Structured Query Language
11) UAT – User Acceptance Testing
12) UI – User Interface
13) Xpath – XML Path Language
14) XML – EXtensible Markup Language
15) CSS – Cascading Style Sheets
16) BPA – Blue Prism Application
17) GUID – Globally Unique Identifier
18) PyPI – Python Package Index
19) HTTPS – HyperText Transfer Protocol Secure
20) UX – User Experience
21) ID – Identifier
22) БД – База Даних
23) CPU – Central Processing Unit
24) RAM – Random Access Memory
25) ІТ – Інформаційні технології
26) ОЗП – Оперативна Запам’ятовуюча Пам’ять
27) SSD – Solid State Drive

ВСТУП

Світова та українська динаміка останніх років демонструє стрімке зростання сегменту онлайн-продажу продуктів харчування. Пандемія COVID-19 остаточно перевела великий відсоток споживачів у режим онлайн-покупок, а роздрібні мережі значно розширили свій асортимент. Месенджери стали для користувачів звичним інструментом взаємодії із сервісами. Одним із найпопулярніших в Україні є Telegram, адже станом на 2025 рік він налічує понад 1 млрд щомісячних активних користувачів в усьому світі [1], а в Україні кількість завантажень перевищує 10,8 млн [2]. Крім того, відкритий API робить Telegram ідеальною платформою для створення сервісних ботів.

Актуальність теми. Бізнес сьогодні стикається з обмеженим бюджетом та необхідністю мінімізувати людські помилки в рутинних операціях. Роботизована автоматизація процесів (RPA) дозволяє швидко інтегрувати нові бізнес-процеси без глибокого втручання в бекенд. Платформа Blue Prism — лідер галузі, що забезпечує масштабоване й контрольоване виконання регламентованих дій у браузері та з файлами. Згідно з даними **Gartner Peer Insights**, її застосовують понад 2 000 компаній у 170 країнах із середньою оцінкою 4,4/5 на основі 800+ відгуків.

Поєднання клієнтського інтерфейсу Telegram-бота та автоматизації Blue Prism дозволяє створити інформаційно-допоміжну систему, яка скорочує шлях від вибору страви до оформлення замовлення. Користувач обирає в боті назву страви — бот формує перелік інгредієнтів і пропонує їх додати до кошика. Підтверджене замовлення потрапляє у файл Excel, звідки утиліта передає його в чергу Blue Prism. Робот перевіряє наявність товарів, додає їх у кошик, оформлює замовлення та фіксує основні кроки у вигляді тегів.

Мета роботи полягає у створенні та дослідженні системи, яка зможе скоротити час формування онлайн-замовлень приблизно на 40–50%.

Для досягнення мети необхідно виконати такі завдання:

– провести детальний аналіз предметної області;

– проаналізувати наявні рішення на ринку онлайн-продуктових сервісів, зокрема популярні сервіси й ботів;

– визначити функціональні та нефункціональні вимоги;

– розробити модель предметної області, включаючи діаграми, які описують взаємодію користувачів із Telegram-ботом та RPA-системою;

– реалізувати Telegram-бота, що забезпечить простий і швидкий пошук страв та інгредієнтів;

– створити процес і бізнес-об’єкт у Blue Prism;

– провести тестування.

Об’єктом дослідження є процес формування та обробки онлайн-замовлення продуктів у торговельній мережі — від вибору страви до підтвердження покупки. **Предметом** — методи й засоби інтеграції клієнтського інтерфейсу Telegram із RPA-платформою для автоматизації цього процесу. **Методична база** ґрунтується на системному аналізі, нотаціях UML, принципах RPA та інструментах Python, Telegram Bot API, Microsoft Excel і Blue Prism.

Структура роботи. У першому розділі виконується системний аналіз предметної області: ставиться задача, оглядаються інформаційні джерела й існуючі рішення, моделюються ключові сценарії взаємодії за допомогою діаграм прецедентів, активностей та послідовностей.

У другому розділі розглядається інформаційне забезпечення: формат і структура recipes.json, роль Excel-файлу як проміжного сховища, архітектура черги Blue Prism з описом полів, SLA, станів і пріоритетів.

Третій розділ присвячено прикладному програмному забезпеченню: реалізації Telegram-бота на Python (модулі config.py та MyOrderBot.py), обробці запитів і передачі в чергу, а також розробці бізнес-об’єкта й процесу в Blue Prism з налаштуванням Control Room.

У четвертому розділі наведено рекомендації щодо впровадження та тестування системи.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання

Інформаційно-допоміжна система для підбору та замовлення продуктів має забезпечити автоматизацію й оптимізацію процесу вибору та оформлення товарів за допомогою Telegram-бота та RPA-технології Blue Prism. Користувачами системи є фізичні особи, які формують замовлення продуктів через мобільний месенджер Telegram.

Основні цілі розробленої системи:

– спростити процес підбору та замовлення продуктів;

– зменшити час, необхідний для оформлення та обробки замовлення;

– автоматизувати взаємодію з онлайн-майданчиком торговельної мережі «Сільпо».

Основні функції, які має виконувати інформаційно-допоміжна система:

– пошук страв та їх інгредієнтів за назвою;

– автоматичне формування списку необхідних продуктів для обраних користувачем страв;

– підтвердження замовлення через інтерфейс Telegram-бота;

– збереження інформації про підтверджені замовлення у файл Excel;

– автоматичне додавання замовлень у робочу чергу Blue Prism;

– автоматичне оформлення замовлення товарів на сайті;

– логування всіх дій, що здійснюються під час обробки замовлення.

Система має містити каталог рецептів страв (у форматі JSON), перелік інгредієнтів, необхідних для приготування кожної страви, список товарів, замовлених користувачем, та статуси обробки замовлень.

Система повинна забезпечувати швидкий і зручний пошук страв і продуктів у Telegram-боті, формування замовлень на основі вибраних рецептів,

можливість перегляду та моніторингу стану замовлення, а також простоту у використанні з мінімальною потребою в додаткових інструкціях.

1.2 Огляд інформаційних джерел та існуючих рішень

За даними досліджень українського ринку e-grocery, у другому півріччі 2024 року виручка в категорії продуктів зросла на 53 % у гривні та на 41 % у доларах США завдяки збільшенню трафіку та кращим конверсіям. Попри загальноукраїнське зростання сегмента, він усе ще характеризується фрагментарною пропозицією: більшість платформ не поєднують підбір за рецептами з оформленням онлайн-замовлень.

Найпопулярнішими сервісами в Україні на сьогодні є **Zakaz.ua**[3], **Glovo** та **Fozzy**[4].

Таблиця 1.1 Порівняльна характеристика e-grocery сервісів в Україні

Сервіс	**Переваги**	**Недоліки**
Zakaz.ua	Широка мережа супермаркетів та магазинів (Metro, Novus, Ашан тощо), онлайн-оплата, акції та знижки, інтуїтивний веб- і мобільний інтерфейс	Немає автоматичного формування замовлення за обраною стравою, відсутня Telegram-інтеграція
Glovo	Швидка доставка, можливість доставки як з магазинів, так і з ресторанів	Асортимент часто обмежений, додаткові збори за експрес-доставку

Таблиця 1.1 Порівняльна характеристика e-grocery сервісів в Україні(закінчення)

Fozzy	Власні склади та логістика, щоденне поповнення, мережа «Сільпо», «Фора», Fozzy Cash & Carry	Працює переважно у найбільших містах (Київ, Львів, Одеса)

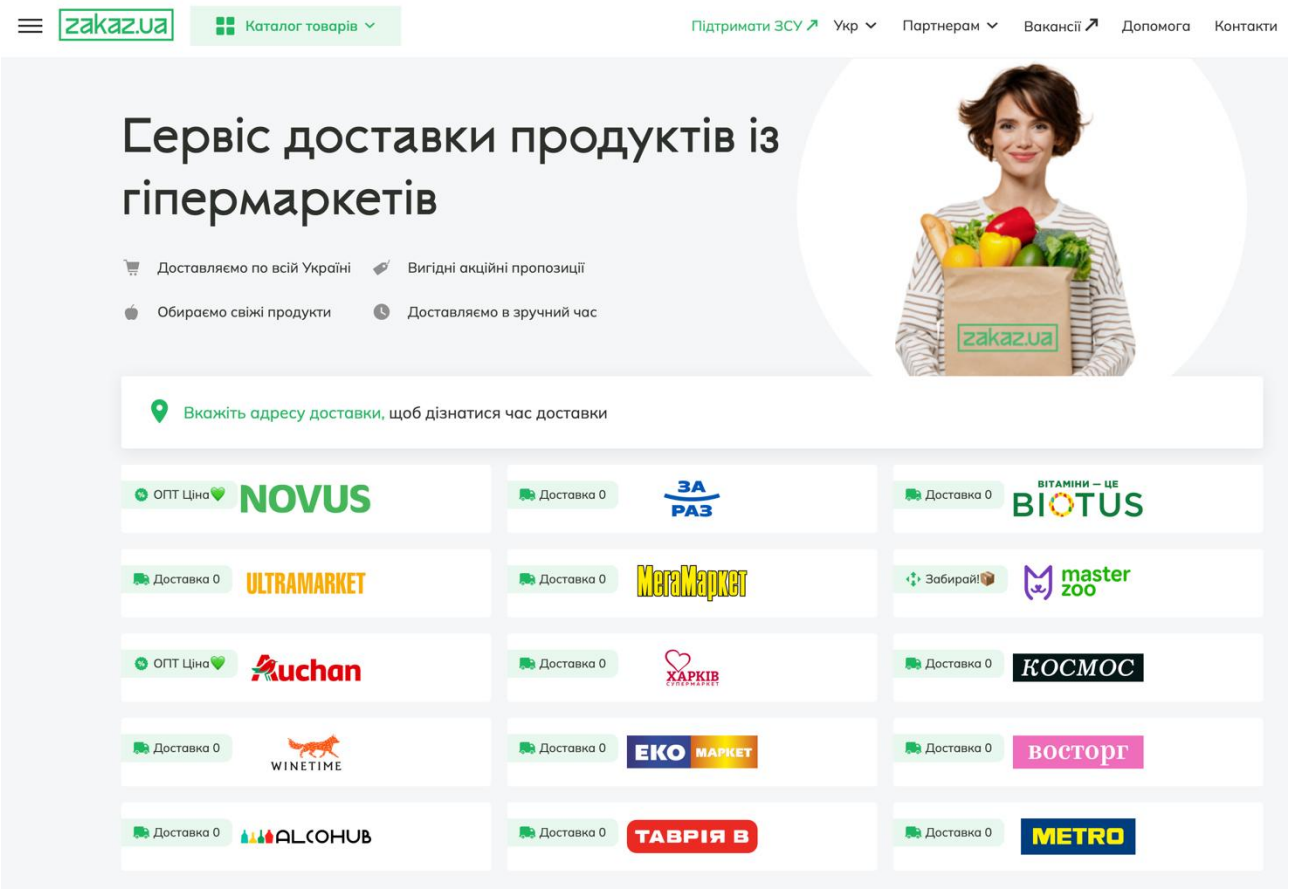


Рис 1.1 Головна сторінка Zakaz.ua

Рис 1.2 Мобільний інтерфейс Glovo

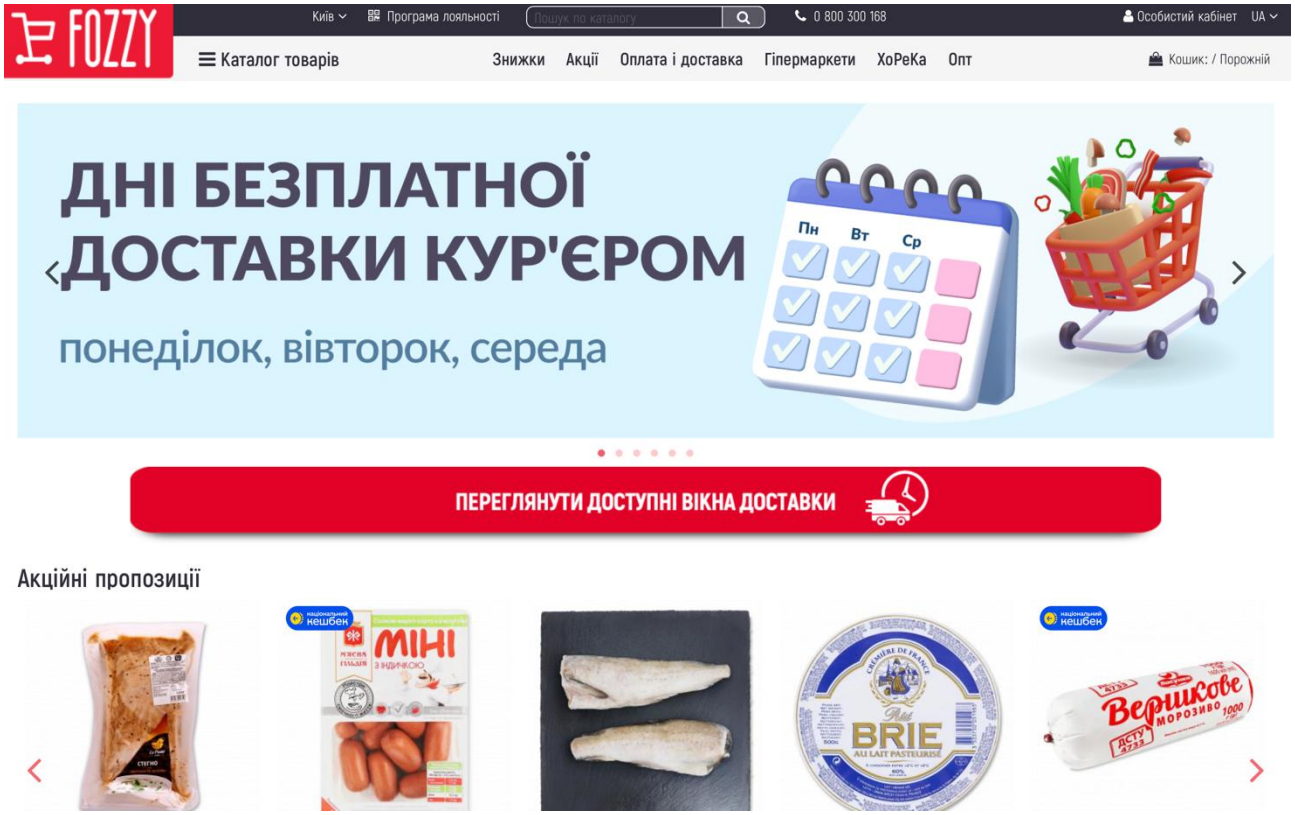


Рис 1.3 Головна сторінка Fozzy

Попри те, що кожна з платформ успішно закриває свою нішу, жодна з них не пропонує:

– автоматизований підбір товарів за рецептурою;
– інтеграцію з Telegram, що працює в реальному часі;
– повну відмовостійкість без участі оператора.

Саме тому створення рішення, яке поєднує зручний чат-інтерфейс Telegram-бота та автоматизовані процеси Blue Prism, відкриває нові можливості: користувач отримує список інгредієнтів в один клік, а RPA-робот на стороні інтернет-магазину гарантує оформлення замовлення з урахуванням усіх побажань і без помилок.

1.3 Моделювання предметної області

Для комплексного вивчення предметної області та точного формулювання вимог до системи було побудовано три взаємопов’язані UML-діаграми: **Use Case**, **Activity** та **Sequence**.

На **діаграмі Use Case** «рис. 1.4» зображено головного актора — **Користувача**, а також ключові варіанти використання: створення списку продуктів, змінення та видалення списку, вказання страви для приготування, отримання списку інгредієнтів, додавання інгредієнтів до списку та підтвердження списку продуктів. Зв’язок включення (include) поєднує варіант Отримання списку інгредієнтів із двома попередніми — Створення списку та Вказання страви, що підкреслює: перед формуванням продуктового набору обов’язково виконується пошук і завантаження інгредієнтів із JSON-файлу. Ця діаграма чітко ілюструє межі системи — що виконує користувач, а що передається на обробку автоматизованим модулям.

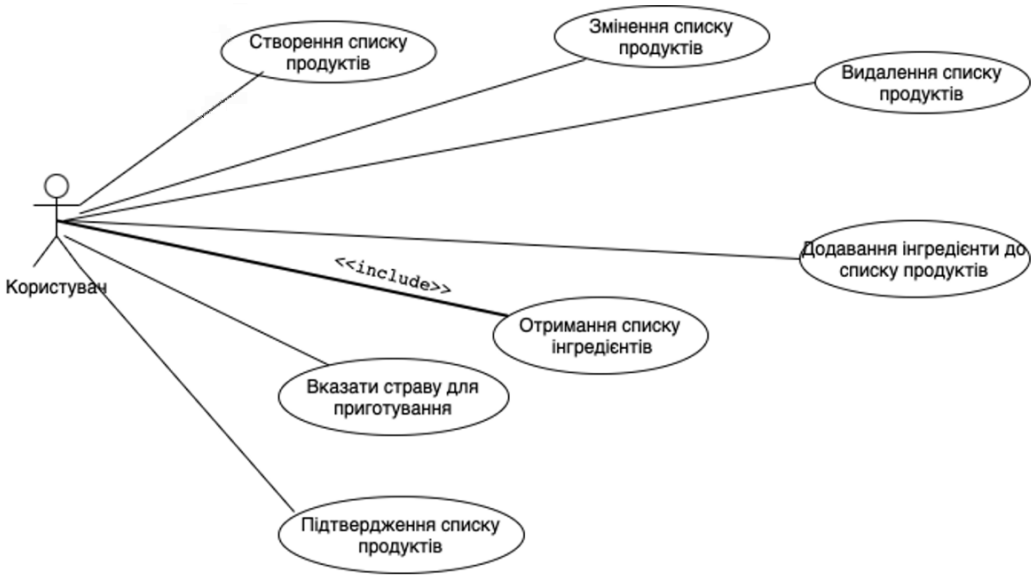


Рис 1.4 Діаграма Use Case

Діаграма активності (Activity) «рис. 1.5» деталізує внутрішню логіку сценарію Створення та підтвердження списку продуктів. Потік починається з ініціалізації роботи бота, далі відбувається перевірка: чи створено список (якщо ні — користувач вводить перші продукти; якщо так — відбувається перехід до вибору страви). Після введення назви страви бот надсилає запит до модуля обробки JSON і отримує перелік інгредієнтів. Користувач може додавати ці елементи до списку, редагувати чи видаляти їх, після чого обов’язково

підтверджує готовий набір. На завершальному етапі бот передає дані до черги Blue Prism. Розгалуження умовних переходів («так» чи «ні») дозволяє обробляти ситуації, коли користувач змінює рішення або вносить корективи.

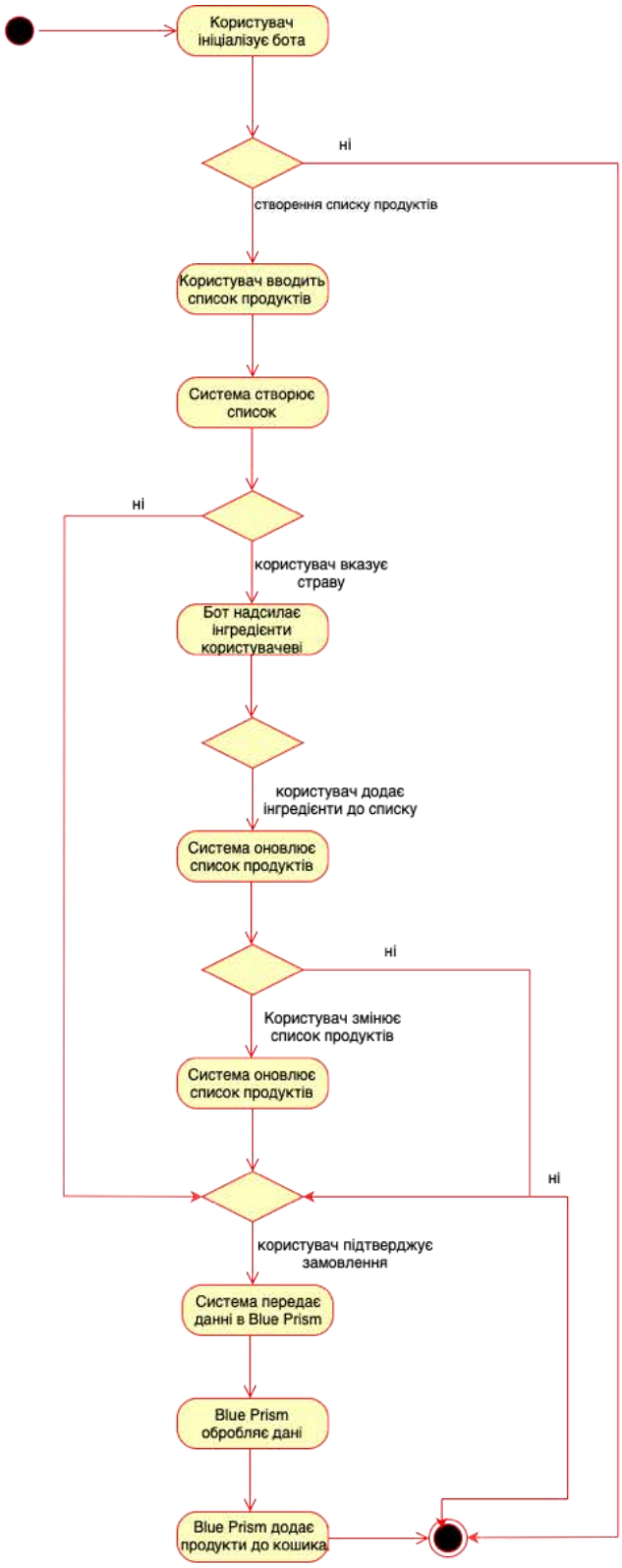


Рис 1.5 Діаграма активності

Діаграма послідовності (Sequence) «рис. 1.6» демонструє взаємодію п’яти об’єктів у часі: Користувач, Telegram-бот, Модуль обробки JSON, Утиліта експорту в Excel та Blue Prism. Спочатку користувач надсилає команду створення списку — бот ініціалізує внутрішню структуру даних і повертає підтвердження. Потім бот отримує назву страви та викликає модуль парсингу, який повертає масив інгредієнтів. Далі бот формує оновлений список і, після підтвердження користувачем, передає інформацію в утиліту, що записує дані до файлу orders.xlsx. Нарешті, утиліта викликає API Blue Prism для додавання транзакції в чергу, а Blue Prism обробляє її асинхронно. Головні повідомлення та відповіді зображені стрілками, що дозволяє простежити, як дані проходять від одного компонента до іншого, а також де можливі часові затримки чи помилки.

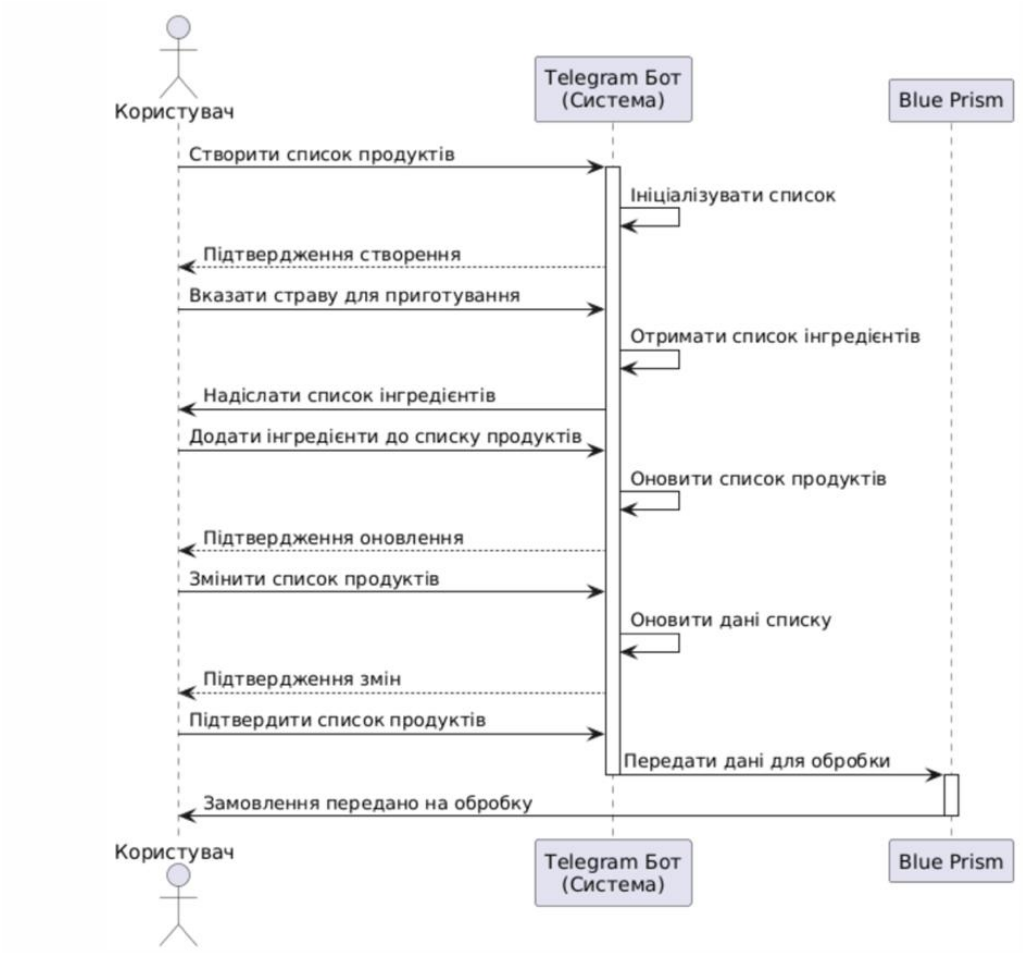


Рис 1.6 Діаграма послідовності

У сукупності ці три діаграми формують цілісне уявлення про систему: вони демонструють, як відбувається взаємодія з користувачем (**Use Case**), як

організована внутрішня логіка (**Activity**) та як реалізовано обмін повідомленнями між компонентами (**Sequence**). Такий багатовимірний підхід гарантує, що всі вимоги до функціоналу й часових залежностей буде враховано ще на етапі проєктування, реалізації та тестування.

1.4 Функціональні та нефункціональні вимоги

Проєктування інформаційної системи потребує чіткого визначення вимог, які мають задовольняти очікування користувачів і враховувати технічні обмеження. У таблицях «1.2» та «1.3» нижче наведено ключові функціональні та нефункціональні вимоги до системи.

Таблиця 1.2 Функціональні вимоги

№	**Функціональні вимоги**
1	Користувач вводить або обирає назву страви в Telegram-боті та отримує перелік необхідних інгредієнтів
2	До підтвердження замовлення користувач може додавати або вилучати товари — бот одразу оновлює загальний список
3	Після команди «Підтвердити» інформація зберігається у файл orders.xlsx і автоматично створює елемент у черзі Blue Prism зі статусом Pending
4	Робот зчитує елемент черги, переходить на сайт silpo.ua[5], шукає зазначені товари та додає їх до кошика
5	Після успішного оформлення замовлення надсилається сповіщення на електронну пошту користувача (Gmail)
6	Усі ключові дії логуються у базі даних Blue Prism для подальшого моніторингу та аналізу
7	У разі тимчасових збоїв робот виконує до трьох повторних спроб; при критичних помилках замовлення переводиться у статус Exception і користувач отримує відповідне повідомлення

Таблиця 1.3 Нефункціональні вимоги

№	**Нефункціональні вимоги**
1	Час відповіді бота не більше 4 с
2	Повний цикл RPA-оформлення одного замовлення (до 10 позицій) не більше 4 хв

Таблиця 1.3 Нефункціональні вимоги (закінчення)

3	Доступність бота 99 % протягом місяця
4	Обмін даними між ботом і сервером здійснюється через захищений HTTPS/TLS-канал
5	Усі основні дії доступні через меню-клавіатуру Telegram
6	Інтерфейс і повідомлення бота українською мовою
7	Не більше трьох кроків від старту бота до підтвердження замовлення
8	Підтримка Telegram на платформах iOS v8.0+, Android v8.0+ та Desktop v4.0+
9	Код бота відповідає стандарту PEP 8, покриття юніт-тестами не менше 70 %

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Формат і структура файлу recipes.json

Для зберігання опису страв і переліку інгредієнтів використовується файл у форматі JSON, що відповідає стандарту RFC 8259 (The JavaScript Object Notation Data Interchange Format).

JSON є легковаговим, текстово-орієнтованим, незалежним від мови програмування форматом обміну даними, який визначає набір правил для представлення структурованих даних у вигляді пар «ключ–значення», масивів і вкладених об’єктів.

У нашому файлі recipes.json дані організовані як масив об’єктів, де кожен об’єкт має таку структуру:

– title (string) — назва страви;

– ingredients (array[string]) — список інгредієнтів.

Приклад фрагмента файлу recipes.json наведено в лістингу 2.1

Лістинг 2.1

```
[
  {
    «title»: «Роли Філадельфія»,
    «ingredients»: [«рис для суші», «норі», «сир філадельфія», «лосось», «огірок»]
  },
  {
    «title»: «Роли Каліфорнія»,
    «ingredients»: [«рис для суші», «норі», «крабові палички», «авокадо», «майонез»]
  }
```

Файл recipes.json завантажується в оперативну пам’ять на етапі ініціалізації Telegram-бота. Після отримання від користувача назви страви бот

виконує пошук відповідного об’єкта за полем title та повертає значення масиву ingredients для подальшого формування замовлення.

Така структура даних є достатньо гнучкою, щоб розширювати каталог новими стравами без необхідності внесення змін до програмного коду.

Повний вміст файлу recipes.json наведено у додатку А.

2.2 Excel-файл orders.xlsx як проміжне сховище даних

У рамках інформаційно-допоміжної системи для підбору та замовлення продуктів файл orders.xlsx виконує роль простого та зручного проміжного сховища замовлень перед їх передачею на обробку платформі Blue Prism.

Наразі структура робочого листа відповідає такому формату «рис 2.1»:

	A	B	C	D	E
1	**User ID**	**Назва**	**Кількість**		
2	504672207	Куряче філе	1		
3	504672207	Пепсі 0.5	5		
4	504672207	Сир Голандський	0,3		
5					
6					
7					

Рис 2.1 Фрагмент файлу orders.xlsx із підтвердженим замовленням

Опис полів:

– **User ID** — унікальний ідентифікатор користувача Telegram, який запустив бота;

– **Назва** — назва товару, доданого до замовлення;

– **Кількість** — числове значення, що вказує на вагу або кількість одиниць товару.

Після підтвердження замовлення Telegram-бот записує для кожного елемента окремий рядок у файл orders.xlsx.

Файл легко зчитується платформою Blue Prism: робот послідовно проходить усі рядки, створюючи для кожної позиції елемент черги Work Queue.

Переваги такого підходу полягають у можливості динамічно додавати, видаляти або редагувати рядки прямо в Excel, а також у простоті інтеграції з Blue Prism, оскільки стандартні дії працюють без необхідності додаткових налаштувань.

2.3 Структура БД Blue Prism (SQL Server) і робоча черга

Blue Prism використовує Microsoft SQL Server як основне сховище для всіх компонентів RPA-платформи. Під час інсталяції створюється кілька баз даних (BPMD, BPAudit, BPRelease тощо), одна з яких відповідає за робочі черги (Work Queues).

За реалізацію черг відповідають такі ключові таблиці:

1) BPAWorkQueue

Містить запис про кожну налаштовану чергу (Work Queue):

- WorkQueueID — унікальний ідентифікатор черги;
- Name — назва черги, наприклад GroceryOrdersQueue;
- Description, MaxSize тощо — додаткові параметри черги.

2) BPAWorkQueueItem

Зберігає елементи черги:

- WorkQueueItemID — унікальний ідентифікатор запису;
- WorkQueueID — зовнішній ключ до таблиці BPAWorkQueue;
- ItemData — JSON-рядок із даними замовлення (наприклад, User ID, Ingredients тощо);
- Priority — пріоритет обробки;
- Status — стан елемента (Pending, Locked, Completed, Exception);
- DateAdded, DateLocked, DateCompleted — часові мітки етапів обробки;
- TimesRetried — кількість повторних спроб.

3) BPAWorkQueueFilter

Надає можливість створювати фільтри для процесів, які зчитують чергу (наприклад, обробка лише елементів із певним пріоритетом).

4) BPAWorkQueueItemTag

Дозволяє додавати до елементів черги теги (наприклад, NotFound — для товарів, які не знайдено).

5) BPAWorkQueueLog

Виконує логування змін станів елементів у черзі: кожна зміна статусу (Status), блокування (Locked), повернення до обробки (Pending) фіксується з часовою міткою.

6) BPMIQueueSnapshot та BPMIQueueTrend

Використовуються для збору аналітики та статистики щодо черги: кількість нових елементів, середній час обробки, продуктивність тощо.

Життєвий цикл елемента черги. Нижче наведено основні етапи обробки кожного елемента черги в Blue Prism — починаючи від моменту створення запису Telegram-ботом до завершення роботи робота:

1) Додавання

Коли користувач підтверджує замовлення в Telegram-боті, бот формує структуру даних (DataTable-колекцію) з полями UserID, Ingredient та Quantity і створює новий запис у таблиці dbo.BPAWorkQueueItem зі статусом Pending та міткою часу DateAdded.

2) Блокування

Процес у Blue Prism виконує дію Get Next Item, яка:

- вибирає найстаріший елемент зі статусом Pending (за полями DateAdded і Priority);
- змінює його статус на Locked;
- встановлює поле DateLocked (час блокування).

3) Обробка

Робот виконує кроки оформлення замовлення на вебсайті silpo.ua, використовуючи дані з однорядкової колекції. Для кожного товару виконується

пошук і додавання до кошика. У разі відсутності товару робот додає відповідний тег у таблиці dbo.BPAWorkQueueItemTag (наприклад, NotFound).

4) Завершення

Після успішного виконання всіх кроків робот оновлює статус елемента в dbo.BPAWorkQueueItem на Completed та фіксує час у полі DateCompleted. Якщо під час обробки виникає помилка, статус змінюється на Exception.

Інтеграція та моніторинг. Для забезпечення надійного та прозорого управління чергою замовлень у Blue Prism передбачено комплекс механізмів інтеграції з Control Room та інструментів моніторингу, що дозволяють операторам і розробникам оперативно відстежувати стан кожного елемента й вчасно реагувати на можливі проблеми.

– **Control Room** дає змогу в реальному часі стежити за кількістю елементів у кожному статусі (Pending, Locked, Completed, Exception) та переглядати логи змін із таблиці BPAWorkQueueLog.

– **Аналітика SLA** (реалізована через користувацьке поле, наприклад, DueBy у BPAWorkQueueItem) дозволяє виявляти прострочені елементи та сигналізувати операторам.

– **Масштабованість** забезпечується можливістю підключення кількох роботів до однієї черги, оскільки механізм блокування (Locked) у BPAWorkQueueItem гарантує, що кожен елемент обробляється лише одним роботом одночасно.

Таким чином, згадані інструменти інтеграції та моніторингу формують єдину платформу контролю якості обробки замовлень, що є критично важливим для підтримки високих SLA-показників і безперебійної роботи системи.

2.4 Проєктування черги (поля, SLA, стани, пріоритети)

У Blue Prism кожен елемент робочої черги передається в процес як **«колекція»** — тип даних, що відповідає типу DataTable у мовах програмування C# та Visual Basic.

Проєктування черги полягає у визначенні структури цієї колекції та налаштуванні відповідних метаданих у таблицях SQL Server.

Приклад полів у колекції (Data Item) для черги GroceryOrdersQueue:

– **User ID** (Number) — унікальний ідентифікатор користувача Telegram;

– **Ingredient** (Text) — назва одного товару;

– **Quantity** (Number) — кількість одиниць у замовленні;

– **Priority** (Number) — пріоритет обробки (елементи з меншим значенням обробляються раніше);

– **DueBy** (DateTime, необов'язкове поле) — крайній термін обробки для реалізації SLA.

Статуси елементів зберігаються у полі Status таблиці BPAWorkQueueItem «рис. 2.2» і можуть набувати таких значень:

– Pending — готовий до обробки;

– Locked — наразі обробляється;

– Completed — успішно виконано;

– Exception — сталася помилка під час виконання.

Пріоритет (Priority) також зберігається у таблиці BPAWorkQueueItem. Він може призначатися ботом або встановлюватися за замовчуванням.

Рис. 2.2 Приклад статусів та пріоритетів елементів черги

Основні таблиці бази даних Blue Prism:

– dbo.BPAWorkQueue — інформація про саму чергу, містить поля WorkQueueID, Name, Description, MaxSize;

– dbo.BPAWorkQueueItem — елемент черги з полями WorkQueueItemID, WorkQueueID (FK), Status, Priority, TimesRetried, DateAdded, DateLocked, DateCompleted;

– dbo.BPAWorkQueueFilter — фільтрація елементів при вибірці;

– dbo.BPAWorkQueueItemTag — додаткові теги для елементів (наприклад, NotFound);

– dbo.BPAWorkQueueLog — історія змін статусів кожного елемента;

– dbo.BPMIQueueSnapshot та dbo.BPMIQueueTrend — таблиці збору статистики й аналітики щодо черги.

У Blue Prism немає вбудованого інструменту контролю SLA (часу виконання завдань), тому було реалізовано **власний механізм**. До кожного елемента черги в полі DueBy записується крайній термін обробки. Окремий моніторинговий процес регулярно перевіряє ці значення, і якщо виявляє, що термін сплив, він позначає замовлення як Exception та надсилає користувачеві повідомлення на Gmail.

Завдяки такому підходу жодне замовлення не залишається «завислим» без обробки, що сприяє дотриманню SLA і підвищує надійність системи.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Реалізація Telegram-бота

3.1.1 Вибір технологічного стеку (Python + aiogram)

Для розробки Telegram-бота з інтеграцією RPA-процесів надзвичайно важливо обрати відповідну мову програмування та фреймворк. Основними критеріями вибору були: швидкість розробки, асинхронна обробка великої кількості запитів, а також широка екосистема бібліотек. Відповідно до цих вимог було обрано мову Python і фреймворк aiogram.

Python — інтерпретована об’єктно-орієнтована мова програмування високого рівня з динамічною типізацією. Вона була розроблена на початку 1990-х років Гвідо ван Россумом[6].

Основні переваги Python:

– читабельність коду (використання відступів замість фігурних дужок робить структуру програми зрозумілішою);

– динамічна типізація та автоматичне керування пам’яттю, що спрощує розробку;

– широка екосистема (тисячі публічних пакетів на PyPI для роботи з базами даних, файлами, HTTP тощо);

– портативність (код на Python працює на різних операційних системах без значних змін).

З огляду на це, Python є ідеальним рішенням для створення Telegram-ботів.

При виборі фреймворку для роботи з Telegram Bot API було розглянуто кілька популярних бібліотек, проте aiogram виявився найкращим варіантом завдяки таким перевагам:

– асинхронна архітектура: aiogram побудований на базі asyncio та aiohttp, що дозволяє обробляти сотні паралельних запитів без блокування основного потоку;

– повна підтримка Telegram Bot API: бібліотека своєчасно оновлюється відповідно до нових версій API і забезпечує доступ до всіх актуальних методів;

– гнучка система хендлерів: дає змогу ефективно керувати логікою обробки повідомлень;

– підтримка inline-меню та клавіатур;

– легка інтеграція з іншими інструментами та модулями Python.

Таким чином, завдяки поєднанню високої продуктивності, гнучкості й простоти масштабування, aiogram став найзручнішим вибором для реалізації Telegram-бота, який повинен бути швидким, адаптивним до діалогів та готовим до обробки значного навантаження.

3.1.2 Архітектура бота та модулі (config.py, MyOrderBot.py)

Архітектура Telegram-бота спроєктована таким чином, щоб забезпечити високу модульність і простоту підтримки програмного рішення. Такий підхід дозволяє ефективно розподілити логіку на окремі незалежні компоненти, кожен з яких відповідає за чітко визначену функцію і може бути легко розширений або змінений без впливу на інші частини системи.

Бот складається з двох основних файлів:

– **config.py** — відповідає за конфігурацію та параметри середовища виконання;

– **MyOrderBot.py** — реалізує бізнес-логіку бота: обробку подій, створення клавіатур, роботу з файлами та передачу даних у систему Blue Prism.

Модуль налаштувань (config.py). Основне призначення модуля config.py полягає в централізованому збереженні параметрів, які можуть змінюватися залежно від середовища, у якому запускається бот. Це суттєво спрощує налаштування системи та підвищує її безпеку завдяки зберіганню конфіденційних даних окремо від основного коду.

У поточній реалізації визначається один параметр — **API_TOKEN**, який виступає унікальним ключем для автентифікації перед Telegram API. Токен отримується за допомогою офіційного бота **BotFather**, що гарантує безпечну

взаємодію із серверами Telegram. Збереження цього токена в окремому файлі дозволяє без зусиль переносити бота між середовищами без потреби змінювати основний код.

Модуль основної логіки (MyOrderBot.py). Файл MyOrderBot.py є основним компонентом Telegram-бота, що реалізує повний життєвий цикл взаємодії з користувачем — від обробки команд до збереження замовлення у файл і передачі його в чергу Blue Prism.

На початку модуля здійснюється імпорт усіх необхідних бібліотек: стандартних модулів Python (asyncio, json, datetime, re, html), бібліотеки pandas для роботи з Excel, а також елементів фреймворка aiogram для взаємодії з Telegram API. У лістингу 3.1 наведено код створення об’єкта Bot та Dispatcher, який керує усією логікою бота.

Лістинг 3.1

```
import asyncio
import json
import os
import re
import html
import pandas as pd
from datetime import datetime
from aiogram import Bot, Dispatcher, F, types
from aiogram.filters import Command
from aiogram.client.bot import DefaultBotProperties
from aiogram.types import (
ReplyKeyboardMarkup, KeyboardButton,
InlineKeyboardMarkup, InlineKeyboardButton
)
import config
bot = Bot(token=config.API_TOKEN,
default=DefaultBotProperties(parse_mode=”HTML”))
```

dp = Dispatcher()

Для забезпечення зручного керування ботом реалізовано два типи клавіатур, які відрізняються способом відображення та функціональністю:

– Reply-клавіатури — відображаються внизу екрана й завжди доступні користувачеві. Приклад реалізації наведено в лістингу 3.2, інтерфейс — на «рис.3.1»

Лістинг 3.2

```
menu_kb = ReplyKeyboardMarkup(
keyboard=[
[KeyboardButton(text=»Меню»)],
[KeyboardButton(text=»Скласти список»)],
[KeyboardButton(text=»Моє замовлення»)],
[KeyboardButton(text=»Інгредієнти»)]
],
resize_keyboard=True
)
```

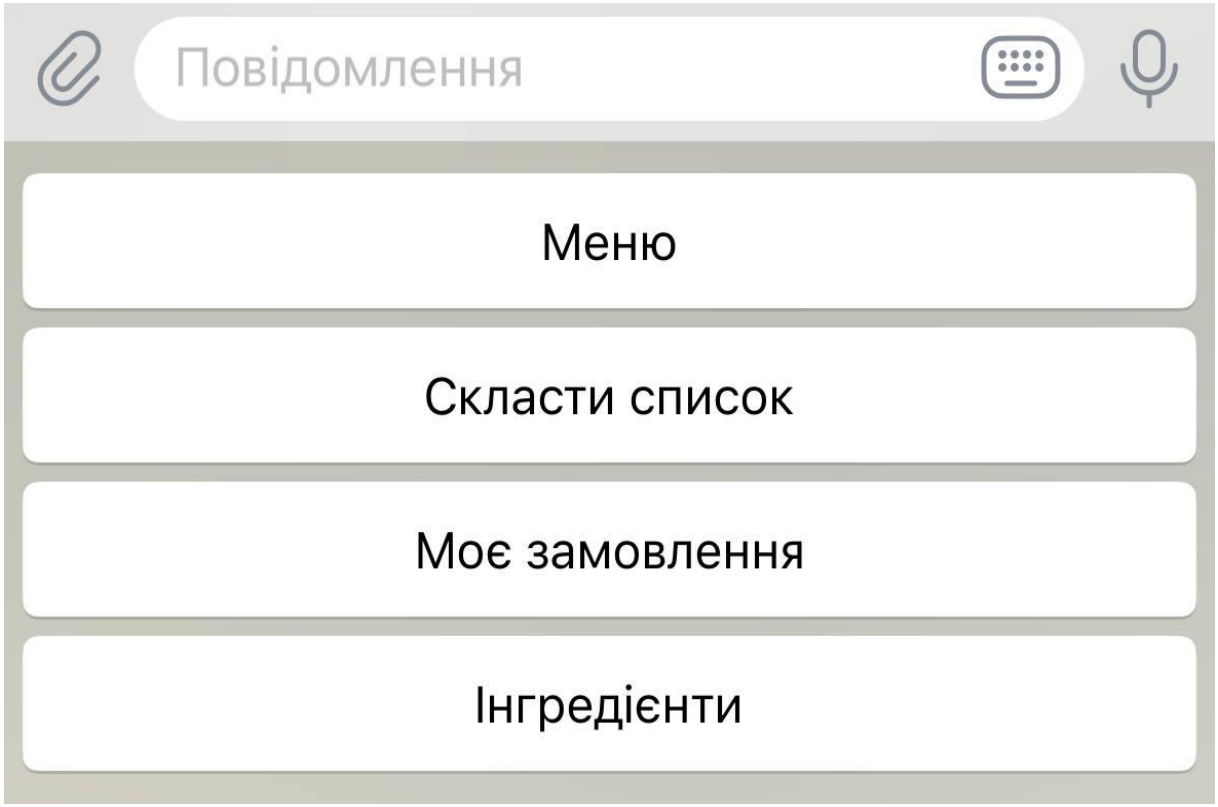


Рис 3.1 Інтерфейс головного меню Telegram-бота з Reply-клавіатурою

– Inline-клавіатури — прив’язуються до окремих повідомлень і використовуються для швидких дій (наприклад, підтвердження або редагування замовлення). Приклад реалізації наведено в лістингу 3.3, інтерфейс — «рис. 3.2».

Лістинг 3.3

```
order_confirm_kb = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text=”Так”, callback_data=”order_confirm_yes”)],
    [InlineKeyboardButton(text=”Змінити”,
callback_data=”order_confirm_change”)],
    [InlineKeyboardButton(text=”Видалити список”,
callback_data=”order_confirm_delete”)],
    [InlineKeyboardButton(text=”Додати елемент”,
callback_data=”order_edit_add”)]
])
```

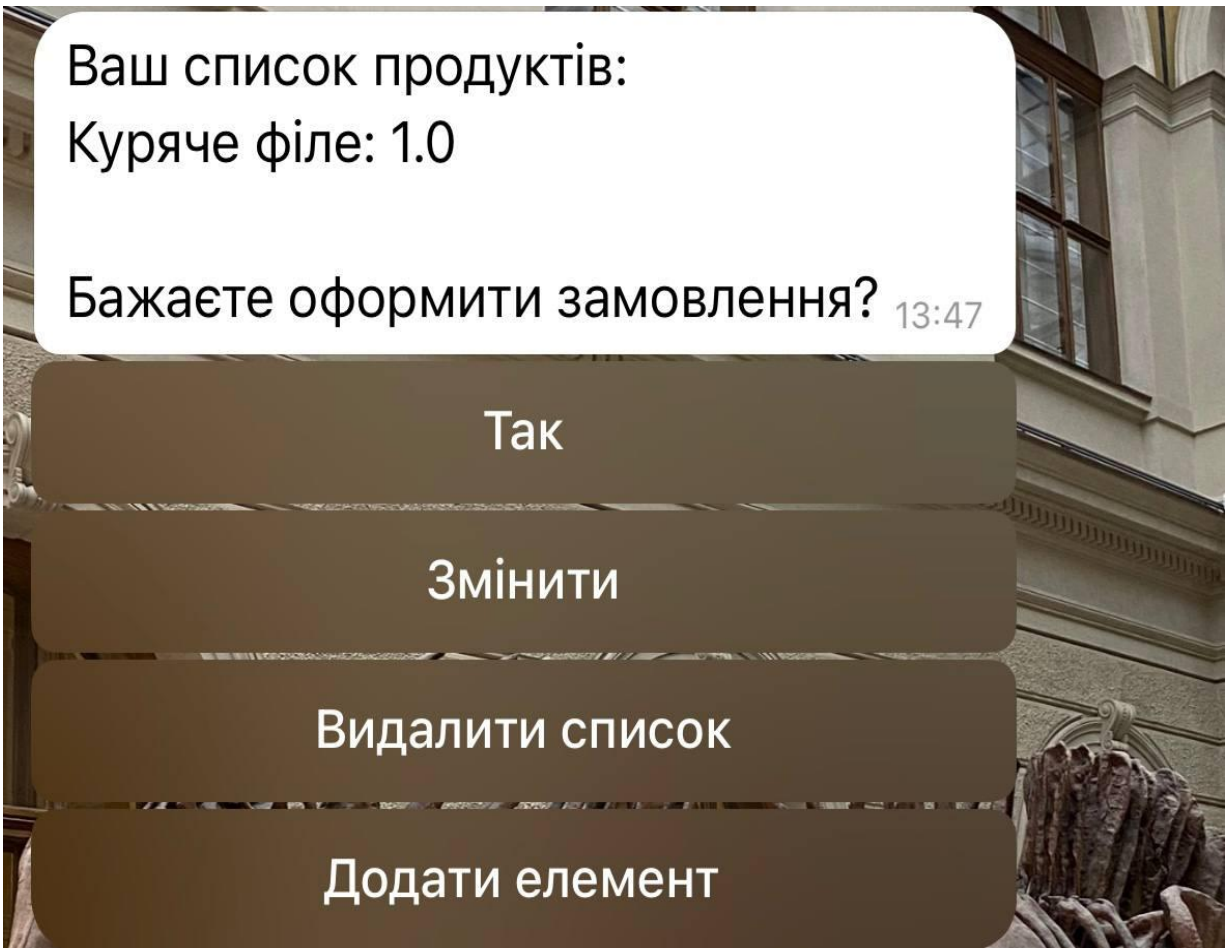


Рис 3.2 Інтерфейс Telegram-бота з Inline-клавіатурою для підтвердження та редагування замовлення

На етапі ініціалізації Telegram-бот зчитує файл recipes.json, який містить каталог страв у вигляді масиву словників. Кожен словник має два поля: title — назва страви, та ingredients — список інгредієнтів, необхідних для її приготування. Завантажені дані зберігаються у змінній recipes, яка використовується для пошуку страв за назвою та автоматичного формування списку продуктів.

Такий підхід забезпечує просте масштабування: нові рецепти можна додавати без зміни коду, достатньо оновити вміст JSON-файлу. Це підвищує гнучкість і знижує залежність бізнес-логіки від структури даних.

Фрагмент коду, який відповідає за завантаження каталогу рецептів, представлено в лістингу 3.4.

Лістинг 3.4

```
with open(“recipes.json”, “r”, encoding=”utf-8”) as f:
   recipes_data = json.load(f)
```

Також у коді реалізовано управління станами користувачів. Для цього використовуються дві змінні:

– **order_states** — фіксує, на якому кроці діалогу перебуває кожен користувач;

– **user_orders** — тимчасово зберігає список інгредієнтів та їх кількість до моменту підтвердження.

Після того як користувач остаточно підтверджує свій список продуктів, бот виконує збереження даних у файл orders.xlsx. Цей крок слугує своєрідним «містком» між Telegram-інтерфейсом і RPA-системою Blue Prism.

Кожен елемент списку (назва товару та його кількість) перетворюється на окремий рядок у таблиці з такими колонками: «User ID», «Назва», «Кількість».

Для роботи з табличними даними використовується бібліотека pandas, яка значно спрощує операції: створення об’єкта DataFrame, додавання нових записів або формування Excel-файлу на основі отриманої інформації.

Фрагмент коду для збереження замовлення в Excel наведено в лістингу 3.5.

Лістинг 3.5

```
def save_order_to_excel(user_id: int, order_list: list):
    rows = [(user_id, name, qty) for name, qty in order_list]
    df_new = pd.DataFrame(rows, columns=[“User ID”, “Назва”, “Кількість”])
    df_new.to_excel(“orders.xlsx”, index=False)
```

Хендлери повідомлень відповідають за обробку введеного користувачем тексту та реакцію Telegram-бота на нього. У модулі реалізовано кілька ключових сценаріїв:

– **Повернення до меню.** При натисканні кнопки «Меню» або введенні відповідного тексту бот очищує внутрішній стан і повертає користувача до стартового інтерфейсу, щоб почати нову сесію взаємодії.

– **Ручне формування списку.** Команда «Скласти список» переводить бота в режим очікування ручного вводу. Користувач вводить перелік товарів у спеціальному форматі, після чого бот обробляє рядок, перевіряє коректність зазначених кількостей і додає кожен елемент до внутрішнього словника user_orders.

– **Перегляд поточного замовлення.** При натисканні кнопки «Моє замовлення» бот витягує з пам’яті всі пари «товар–кількість» та надсилає користувачу текстовий перелік із можливістю подальшого редагування.

– **Запит інгредієнтів за назвою страви**. Якщо користувач вводить назву рецепту, бот шукає її в завантаженому масиві recipes_data. У разі успішного пошуку бот надсилає список інгредієнтів і пропонує додати їх до замовлення; у разі невдачі — просить уточнити запит.

Фрагмент коду для обробки сценарію «Моє замовлення» наведено в лістингу 3.6.

Лістинг 3.6

```
async def view_order_handler(message: types.Message):
    user_id = message.from_user.id
    if user_id not in user_orders or not user_orders[user_id]:
```

```
        await message.answer("Ваш список порожній.",
reply_markup=menu_kb)
    else:
        order_text = "\n".join(
            f"{i+1}. {html.escape(name)}: {qty}"
            for i, (name, qty) in enumerate(user_orders[user_id])
        )
        await message.answer(
            f"Ваш список:\n{order_text}\n\nБажаєте оформити замовлення?",
            reply_markup=order_view_kb
        )
```

У розробці інтерфейсу бота використано callback-хендлери для inline-кнопок. Такі кнопки дають змогу керувати списком замовлення без додаткового введення тексту. Оброблено кілька базових сценаріїв:

– **Підтвердження списку.** Кнопка «Так» запускає експорт даних у Excel, очищує службові словники та надсилає користувачеві підсумкове повідомлення про успішне оформлення замовлення.

– **Редагування елементів.** Дія «Змінити» переводить бота в режим уточнення деталей (наприклад, кількості або назви товару), після чого повертає оновлений список за допомогою тієї самої inline-клавіатури.

– **Додавання нового елемента.** Після натискання кнопки «Додати елемент» бот запитує у чаті назву й кількість товару, обробляє введені дані та додає їх до замовлення.

– **Видалення позицій.** Кнопка «Видалити елемент» прибирає вибраний товар; дія «Видалити весь список» очищує словник замовлення й повертає користувача до меню.

Фрагмент коду, що реалізує підтвердження списку, наведено в лістингу 3.7.

Лістинг 3.7

```
async def order_confirm_callback_handler(callback: types.CallbackQuery):
```

```
    «»»
    Обробник кнопок у дії «Скласти список».
    “””
    user_id = callback.from_user.id
    data = callback.data
    if data == “order_confirm_yes”:
        order_list = user_orders.get(user_id, [])
        save_order_to_excel(user_id, order_list)
        await callback.message.edit_text(“Ваше замовлення оформлене!”,
reply_markup=None)
        user_orders.pop(user_id, None)
        await bot.send_message(user_id, “Повертаємось до головного меню:”,
reply_markup=menu_kb)
        await callback.answer()
```

Кожен callback-хендлер завершує свою роботу методом answer() або edit_message() — це забезпечує коректне закриття вікна повідомлення (popup) та оновлення інтерфейсу без дублювання в чаті. Такий підхід значно покращує UX, оскільки всі дії підтверджуються в межах поточного повідомлення, не «засмічуючи» історію зайвим текстом.

У кінці файлу MyOrderBot.py реалізовано функцію main(), яка відповідає за запуск циклу обробки оновлень.

Точка входу до застосунку наведена в лістингу 3.8.

Лістинг 3.8

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

– main() — асинхронна функція, яка всередині виконує попередні налаштування (наприклад, uvloop.install(), якщо використовується), а також запускає опитування серверів Telegram через dp.start_polling().

– Виклик asyncio.run(main()) створює та запускає подієвий цикл, що забезпечує асинхронну обробку всіх вхідних подій Telegram-бота.

Основний варіант експлуатації бота на Windows-сервері реалізовано через BAT-файл run_bot.bat, який виконує команду python MyOrderBot.py. Це дозволяє запускати бота подвійним кліком або автоматизувати його старт через планувальник завдань Windows. Усі журнали роботи (логи) можуть бути гнучко перенаправлені у зовнішній файл для зручної діагностики.

Повний код файлу MyOrderBot.py наведено в додатку Б.

3.1.3 Пошук страв та формування замовлення (парсинг JSON)

Цей розділ описує процедуру взаємодії користувача з Telegram-ботом під час пошуку страв у внутрішньому каталозі (recipes.json) та формування списку продуктів, необхідних для приготування обраної страви.

Файл recipes.json є основним джерелом даних, що містить назви страв та відповідні інгредієнти, необхідні для їх приготування. Структура файлу організована у вигляді JSON-масиву, де кожен елемент є словником із двома ключами:

– "title" — назва страви (наприклад, «Піца Маргарита»);

– "ingredients" — масив рядків, що представляють інгредієнти для приготування страви (наприклад: ["борошно", "дріжджі", "томатний соус", "моцарела", "базилік"]).

При запуску бота файл із рецептами завантажується в оперативну пам’ять як Python-структура для швидкого доступу до даних. Фрагмент коду завантаження каталогу було показано в лістингу 3.4. Ця операція виконується лише один раз, що дозволяє значно пришвидшити пошук рецептів у подальшій роботі бота.

Після введення користувачем назви страви бот виконує пошук відповідного рецепту серед завантажених даних. Алгоритм пошуку працює так:

спочатку бот отримує повідомлення, що містить назву страви (наприклад, *«піца»*), далі текст перетворюється в нижній регістр для забезпечення

незалежності від регістру, після чого викликається функція пошуку, яка проходить по списку об’єктів і шукає збіги за полем title.

Фрагмент коду пошуку рецепту наведено в лістингу 3.9.

Лістинг 3.9

```
def search_recipes(query: str):
query = query.lower()
return [recipe for recipe in recipes_data if query in recipe[«title»].lower()]
```

Після цього можливі три варіанти розвитку подій:

– Якщо збігів немає, користувачеві повертається повідомлення: «Вибачте, страву не знайдено. Спробуйте інший запит.»

– Якщо знайдено один збіг, бот одразу переходить до формування замовлення, автоматично додаючи інгредієнти до тимчасового списку користувача.

– Якщо знайдено декілька збігів, бот надсилає повідомлення з inline-кнопками, щоб користувач міг обрати потрібну страву, одночасно зберігаючи варіанти у словнику pending_recipes[user_id].

Коли користувач обирає страву, бот отримує список відповідних інгредієнтів і переходить до етапу підтвердження їх додавання до замовлення. Список інгредієнтів тимчасово зберігається у словнику pending_ing_addition[user_id].

Бот надсилає повідомлення із запитанням:

«Чи бажаєте додати ці продукти до вашого списку?»

Разом із цим повідомленням відображаються inline-кнопки:

– «Так» — додає інгредієнти до замовлення (генерує callback-запит add_ing_yes);

– «Ні» — ігнорує список інгредієнтів (генерує callback-запит add_ing_no).

У разі натискання кнопки «Так» інгредієнти зберігаються в основному списку замовлення user_orders[user_id] з кількістю за замовчуванням (1

одиниця). Якщо користувач обирає «Ні», список інгредієнтів ігнорується, і користувач повертається до головного меню.

Після додавання інгредієнтів користувач може переглянути поточний список продуктів. Бот надсилає повідомлення зі сформованим переліком, у якому вказано назви товарів і відповідні кількості. Разом зі списком відображається inline-клавіатура з кнопками:

– «Змінити»
– «Додати елемент»
– «Видалити елемент»
– «Видалити весь список»

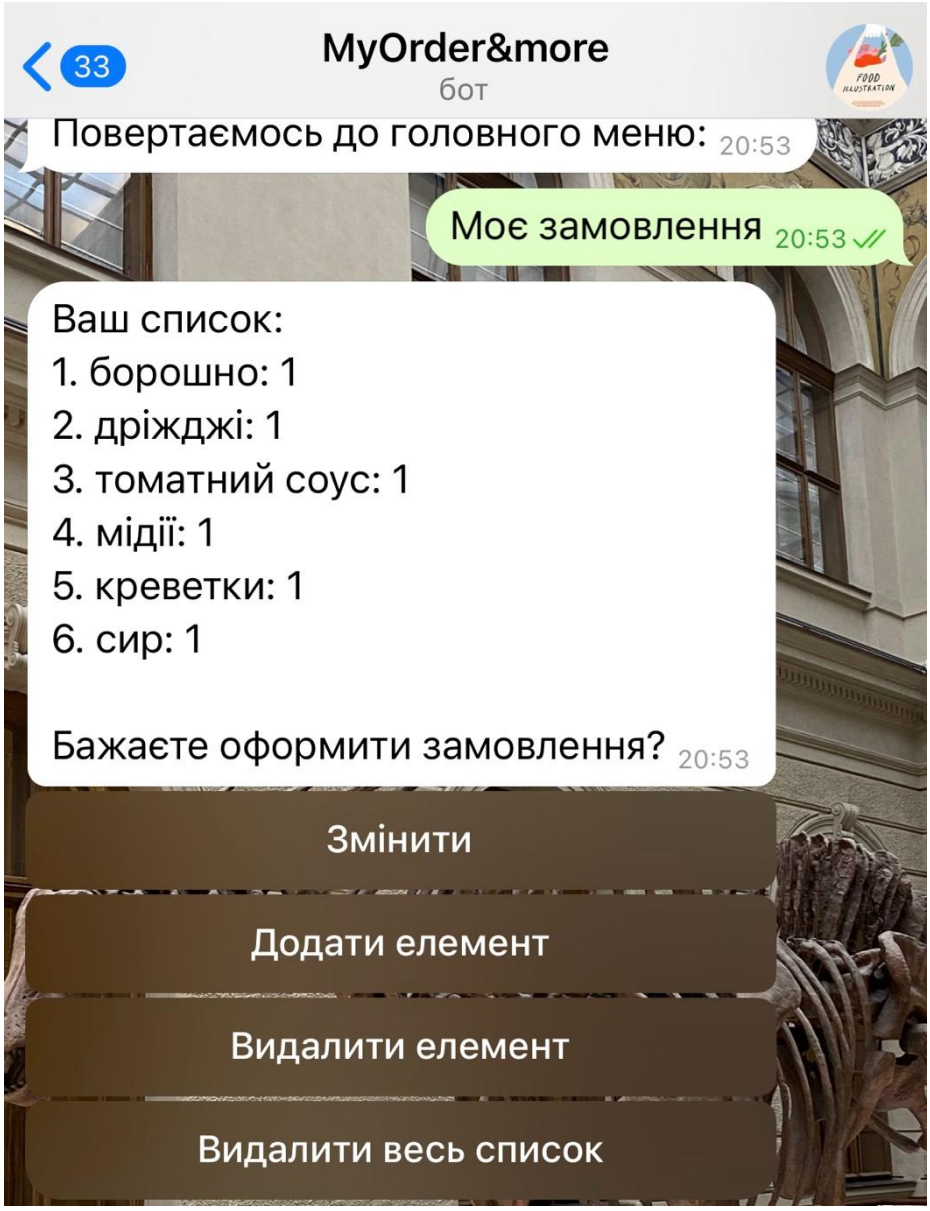


Рис 3.3 Інтерфейс Telegram-бота з Inline-клавіатурою для перегляду замовлення

Кожна з кнопок викликає відповідний callback-хендлер, який реалізує подальшу логіку редагування.

У коді також передбачено обробку помилкових та виняткових ситуацій. Якщо введення користувача було некоректним (наприклад, із помилкою в назві страви), бот повертає пояснення з проханням повторити запит або уточнити назву. Якщо ж страву знайдено, але поле "ingredients" порожнє, бот надсилає повідомлення про те, що для цієї страви інгредієнти не вказані, та пропонує обрати інший варіант.

Окрім того, кожен крок взаємодії супроводжується перевіркою поточного стану користувача, що запобігає випадковому порушенню логіки формування замовлення.

Використання структури JSON дозволяє швидко та ефективно здійснювати пошук страв і відповідних інгредієнтів, а продумана логіка обробки повідомлень і callback-кнопок забезпечує зручність користування й мінімізує ризик помилок.

3.1.4 Експорт замовлення у orders.xlsx та передача до BP-черги

Після того як користувач остаточно підтвердив свій список продуктів, відбувається двоетапний «міст» між Telegram-ботом і платформою Blue Prism:

– створення та запис даних у файл Excel;

– передача інформації до черги Blue Prism за допомогою компонента MS Excel VBO.

Формування Excel-файлу реалізовано у функції save_order_to_excel(user_id, order_list), яка створює табличну структуру для кожного підтвердженого замовлення. Ця таблиця легко зчитується у Blue Prism як колекція.

На першому етапі виконується підготовка даних: кожен кортеж (назва товару, кількість) перетворюється на рядок із додаванням user_id. Далі відбувається запис у файл за допомогою методу to_excel(). Параметр index=False вимикає запис індексу рядків, залишаючи у файлі лише корисні дані.

Фрагмент коду формування Excel-файлу наведено в лістингу 3.10.

Лістинг 3.10

```
def save_order_to_excel(user_id: int, order_list: list):
    rows = [(user_id, name, qty) for name, qty in order_list]
    df_new = pd.DataFrame(rows, columns=[“User ID”, “Назва”, “Кількість”])
    df_new.to_excel(“orders.xlsx”, index=False)
```

Якщо файл уже існує, бот може зчитати попередні записи за допомогою функції pd.read_excel() і додати нові без втрати даних. Файл orders.xlsx зберігається в кореневій директорії проєкту, що дозволяє Blue Prism звертатися до нього без складних абсолютних шляхів.

Після генерації orders.xlsx RPA-робот виконує такі кроки:

– Open Workbook (MS Excel VBO) — відкриває файл orders.xlsx.

– Get Worksheet as Collection — зчитує вміст аркуша у вигляді колекції (однорядковий DataTable у .NET), де кожен запис містить поля User ID, Назва, Кількість. Ця дія є стандартною частиною MS Excel VBO у Blue Prism і дозволяє швидко перетворити табличні дані у формат, зручний для подальшої обробки.

– Loop (For Each) — перебирає колекцію, і для кожного рядка викликає дію Add To Queue (Work Queue VBO), передаючи поля як елемент однорядкової колекції: UserID, Ingredient, Quantity.

– Add To Queue — створює запис у таблиці dbo.BPAWorkQueueItem зі статусом Pending, пріоритетом за замовчуванням і часовою міткою DateAdded. Поля передаються як DataItem до черги, що забезпечує коректну передачу значень у процес.

– Close Workbook — закриває екземпляр Excel і звільняє використані ресурси.

Переваги такого підходу полягають в уніфікованому інтерфейсі, простоті налагодження та високій гнучкості. Оскільки файл Excel є єдиною точкою контакту між ботом і роботом, немає потреби використовувати API або базу даних. Крім того, є можливість вручну відкрити orders.xlsx і перевірити всі записи перед запуском автоматичної обробки.

Розділ 3.1 описує завершальний етап реалізації Telegram-бота, на якому згенеровані ботом дані перетворюються у формат, зрозумілий для RPA-процесу. Такий підхід забезпечує надійну інтеграцію між клієнтським інтерфейсом і системою автоматизації на базі Blue Prism.

3.2 Реалізація RPA-рівня в Blue Prism

3.2.1 Теоретичні основи Blue Prism (архітектура платформи, Process vs Object Studio, Queues)

Blue Prism — це корпоративна платформа роботизованої автоматизації процесів (Robotic Process Automation, RPA), що з 2001 року розвивається британською компанією Blue Prism Ltd [7]. Програмні «роботи» (digital workers), створені в Blue Prism, імітують дії користувача в графічному інтерфейсі прикладних систем, виконуючи рутинні, повторювані завдання без участі людини.

Платформа орієнтована на великі та середні підприємства й підтримує високі вимоги до безпеки, масштабованості та централізованого управління.

Ключові компоненти екосистеми Blue Prism:

Process Studio — візуальне проєктування бізнес-процесів у вигляді flow-діаграм, з багатонитковою логікою, обробкою винятків та гілкуванням.

– **Object Studio** — створення Business Objects (модулів взаємодії з конкретними застосунками) через Application Modeller.

– **Control Room** — диспетчеризація та моніторинг digital workers у реальному часі: запуск, зупинка, планування, перегляд черг.

– **System Manager** — централізовані налаштування користувачів, ролей, доступу, безпеки та облікових даних.

– **Release Manager** — пакетування процесів і об’єктів у релізи з розширенням .bprelease, а також керування версіями.

– **Runtime Resource** — «робоча станція» або сервер без графічного інтерфейсу, на якому виконуються роботи; підтримує горизонтальне масштабування.

– **SQL Database** — єдине сховище конфігурацій, журналів (audit trail), метаданих і черг Work Queue.

Життєвий цикл автоматизованого процесу виглядає наступним чином: спочатку відбувається розробка у Process/Object Studio. Після цього виконується тестування у середовищі UAT, далі – пакетування у .bprelease та впровадження через Release Manager, потім – оркестрація у Control Room: планування, розподіл ресурсів, моніторинг черг. Останній етап – підтримка та масштабування: перегляд логів, оптимізація ботів, оновлення релізів.

Основні переваги Blue Prism – це low-code-підхід, оскільки навіть бізнес-аналітик без глибоких знань програмування може створювати процеси методом drag-and-drop, а також гнучка інтеграція (підтримка як UI-автоматизації, так і API-методів).

Process Studio vs Object Studio. Blue Prism дотримується принципу чіткого розподілу відповідальності між проєктуванням бізнес-логіки (що робити?) та технічною взаємодією з інтерфейсами (як саме це зробити?). Для цього платформа надає дві окремі студії — Process Studio та Object Studio.

Таблиця 3.1 Порівняння Process Studio та Object Studio

Ознака	Process Studio	Object Studio
Призначення	Побудова високорівневого workflow-процесу	Інкапсуляція взаємодії з конкретними застосунками
Основні елементи	Start, End, Action, Decision, Calculation, Loop, Exception	Input, Output, Action, Wait, Read

Таблиця 3.1 Порівняння Process Studio та Object Studio(закінчення)

Рівень повторного використання	Координує виклики різних BO, не містить коду UI-взаємодії	Окремий BO може багаторазово викликатись із різних процесів
Фокус	Логіка бізнес-процесу, керування чергами, обробка винятків, SLA	Технічні дії: клік, введення тексту, читання даних, API-запити
Інструмент UI-спостереження	Відсутній (працює лише з BO)	Application Modeller (Win32, HTML, Java, Citrix, Mainframe)
Типові об’єкти даних	Collections, Variables, Work Queue Items	Local/Global Data Items, Element Attributes
Приклад дії	“Mark Queue Item Completed”	Click Button “Add to Cart”

Структура процесу в Process Studio має наступний вигляд:

Start → основний блок логіки → End. Для кожного етапу бізнес-процесу створюється окремий Page (підпроцес), наприклад: Initialization, Process Item, Close. Main Page виконує обхід елементів Work Queue, викликаючи Process Item у циклі.

Категорії стейджів у Blue Prism поділяються так:

– **Data (Data, Collection, Data Item)** — зберігання тимчасових змінних;

– **Logic (Decision, Choice, Calculation)** — умовні переходи та обчислення;

– **Flow (Anchor, Link)** — структуризація діаграми;

– **Sub-Process / Action** — виклик інших процесів або дій BO;

– **Exception (Recover, Resume, Exception)** — централізована обробка помилок.

Керування винятками реалізується наступним чином:

– Використання шаблону Recover–Resume;

– Логування помилок у таблиці BPAWorkQueueLog із деталізацією Exception Type та Exception Detail;

– Переведення елемента черги у статус Exception або повторна спроба (Retry) із лічильником TimesRetried.

Для кращого сприйняття проєкту рекомендується дотримуватися низки практик: кожен процес має відповідати окремій бізнес-функції; назви процесів слід оформлювати у стилі PascalCase; а на ключових етапах доцільно залишати коментарі у вигляді Stage Notes для документування логіки.

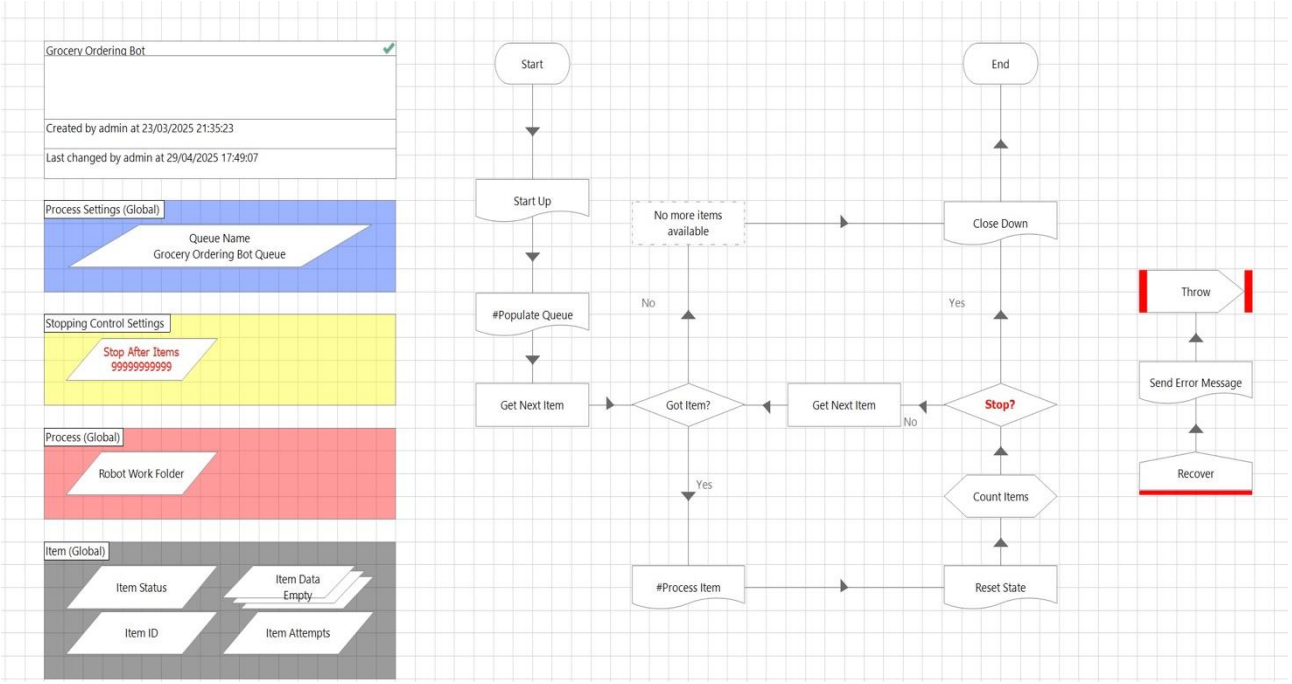


Рис 3.4 Інтерфейс Process Studio з основними елементами (Action, Decision, Loop)

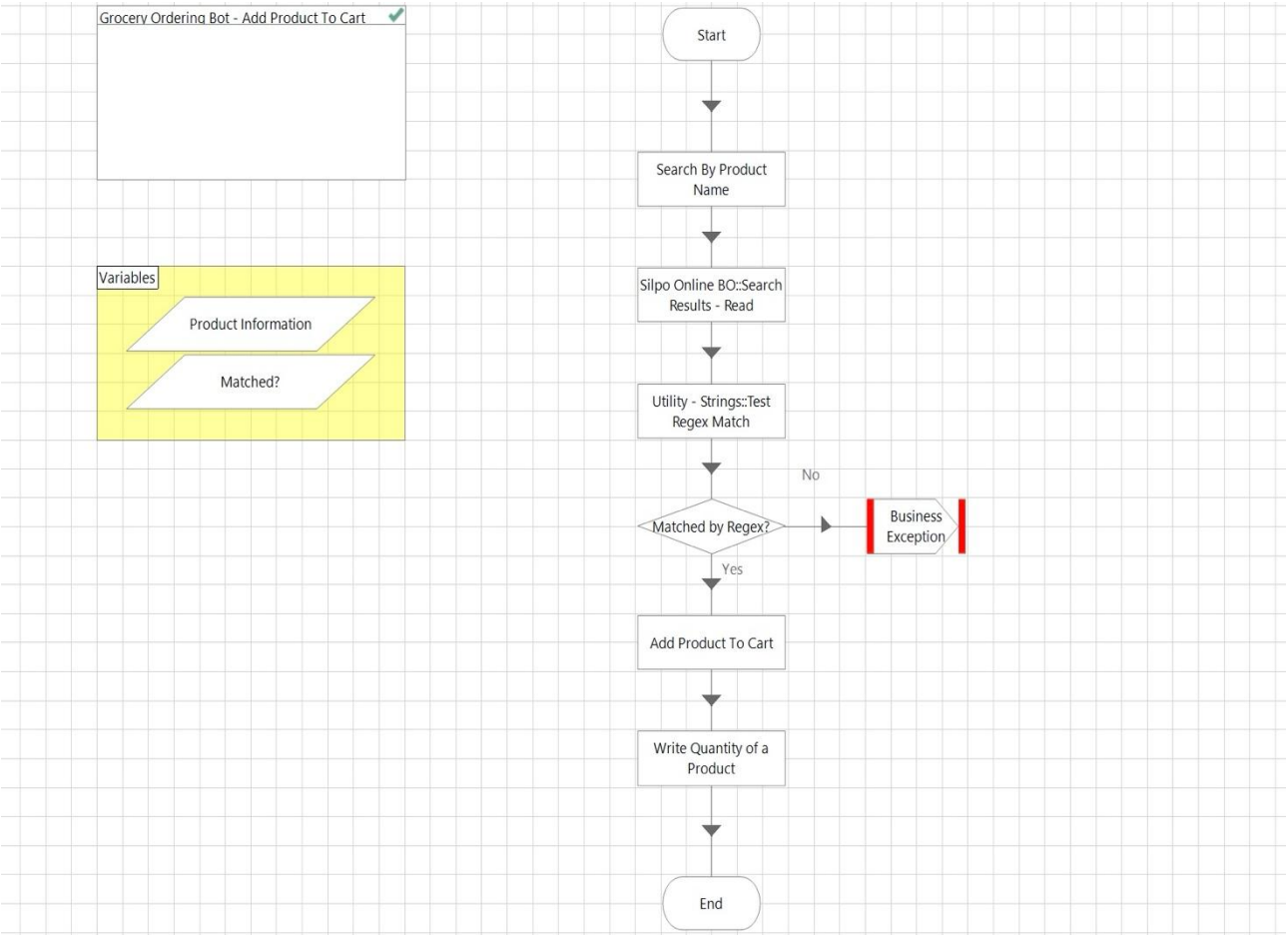


Рис 3.5 Приклад блок-схеми процесу для автоматизації оформлення замовлення

У Object Studio структура Business Object (BO) має такий вигляд:

– Application Modeller описує вікно або сторінку застосунку, а також окремі елементи інтерфейсу (наприклад, за допомогою XPath, CSS-селекторів або атрибутів Win32).

– Actions реалізують логіку взаємодії з UI на прикладному рівні, зокрема дії типу Login, SearchProduct, AddToCart, Checkout.

– Кожна Action може містити власні сторінки-підпроцеси, як-от Navigate «рис. 3.6», Read, Write тощо.

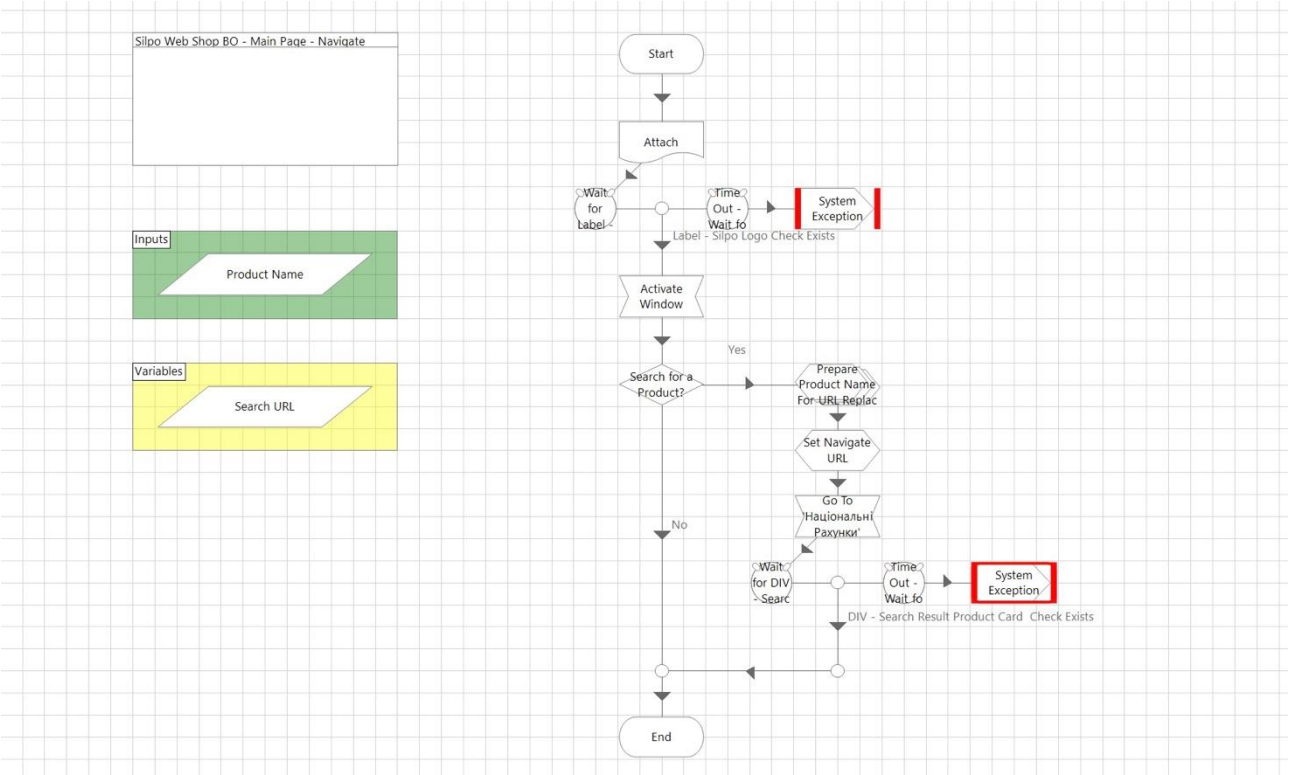


Рис 3.6 Приклад сторінку типу «Navigate»

Також у Blue Prism передбачено кілька режимів «шпигування» (spying modes)[8] — способів виявлення та ідентифікації елементів інтерфейсу користувача під час створення моделі програми в Application Modeller. Кожен режим відповідає конкретному типу технології або середовища, у якому працює цільовий застосунок.

Основні режими spying:

– HTML — для стандартних вебсторінок, що працюють у браузерах.

– Java — для Java-аплетів та інтерфейсів, створених за допомогою Swing або AWT.

– Win32 / Accessibility — для класичних настільних (desktop) застосунків на базі Windows Forms або MFC.

– UIA (UI Automation) — для сучасних інтерфейсів, зокрема UWP та нових версій WinForms.

– Citrix / Surface Automation — для віртуалізованих середовищ, де робота з UI відбувається через координати або зображення елементів.

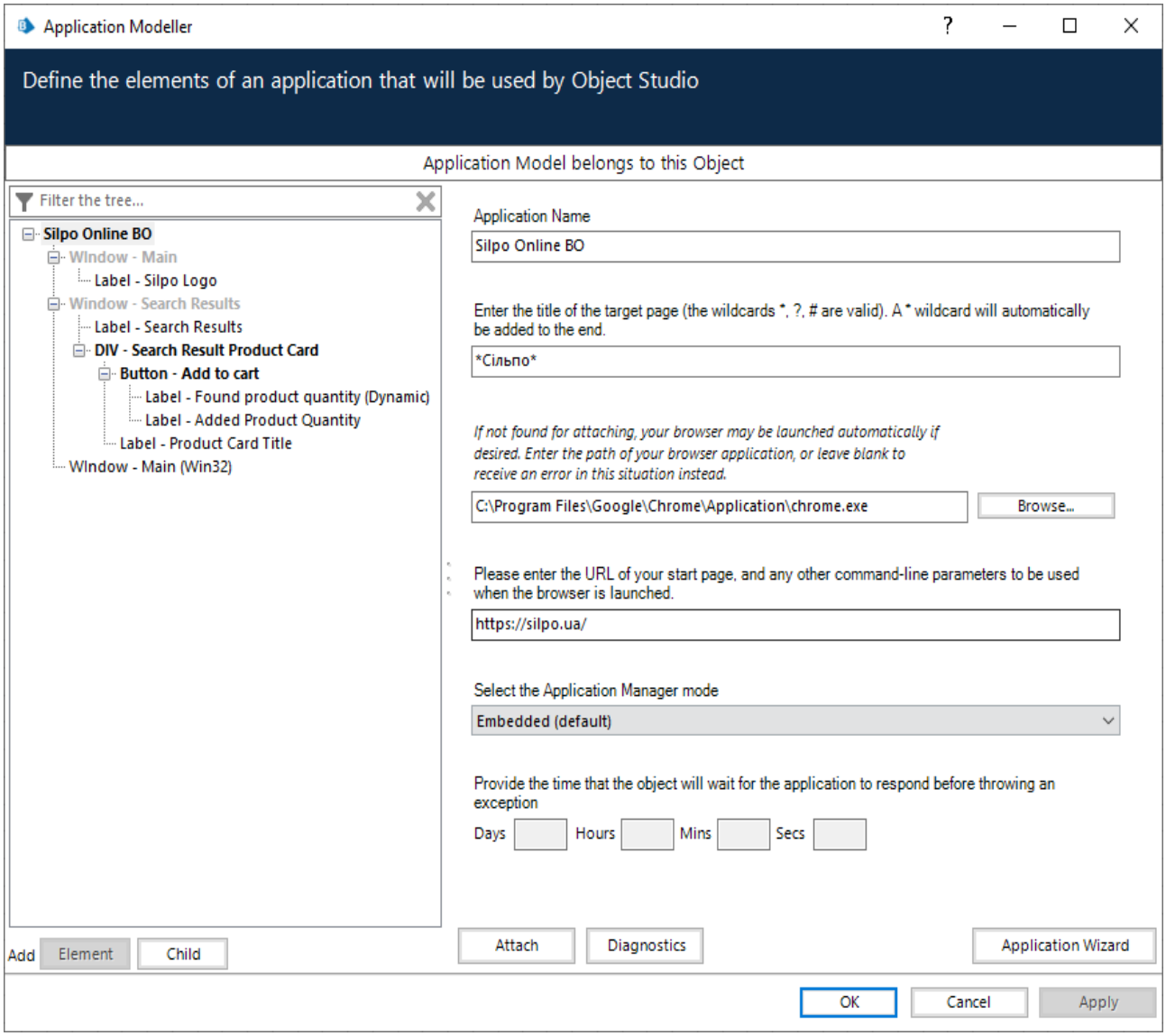


Рис 3.7 Інтерфейс Object Studio (приклад роботи Application Modeller)

У середовищі Blue Prism wait-стратегія — це керований алгоритм «пауза–перевірка», який дає роботові час на коректну взаємодію з динамічним інтерфейсом. Мета такої стратегії — синхронізувати швидкість виконання процесу з реальною швидкістю відгуку цільового застосунку, аби уникнути помилок на кшталт element not found, invalid handle тощо.

Wait-стратегії поділяються на:

– Active Wait — очікування появи елемента. Підходить для діалогових вікон, спливаючих повідомлень, кнопок Submit після валідації.

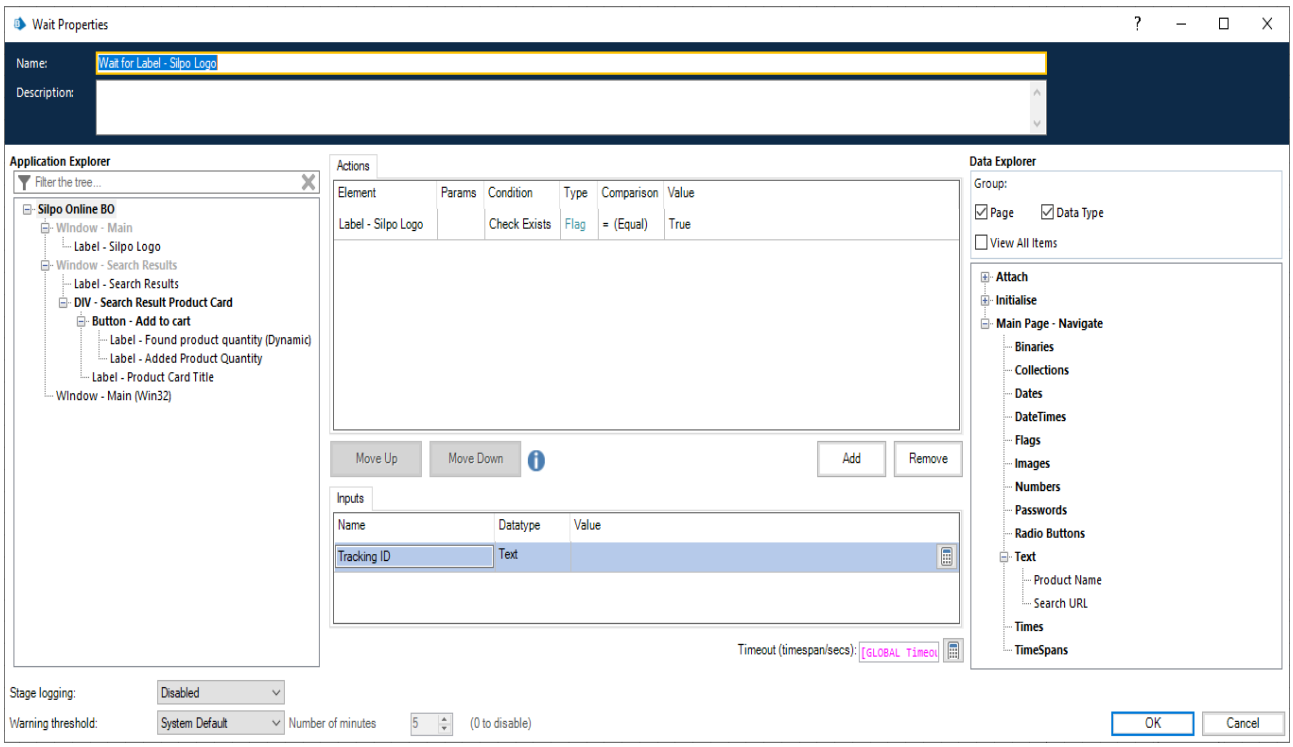


Рис. 3.8 Інтерфейс Wait stage з прикладом Active Wait

– Timeout + Recover із власним Exception Message — дозволяє отримати чіткий лог-трейс для подальшої підтримки та спрощує аналіз інцидентів. Рекомендується обгортати Recover + Resume у окремий блок Exception Handling.

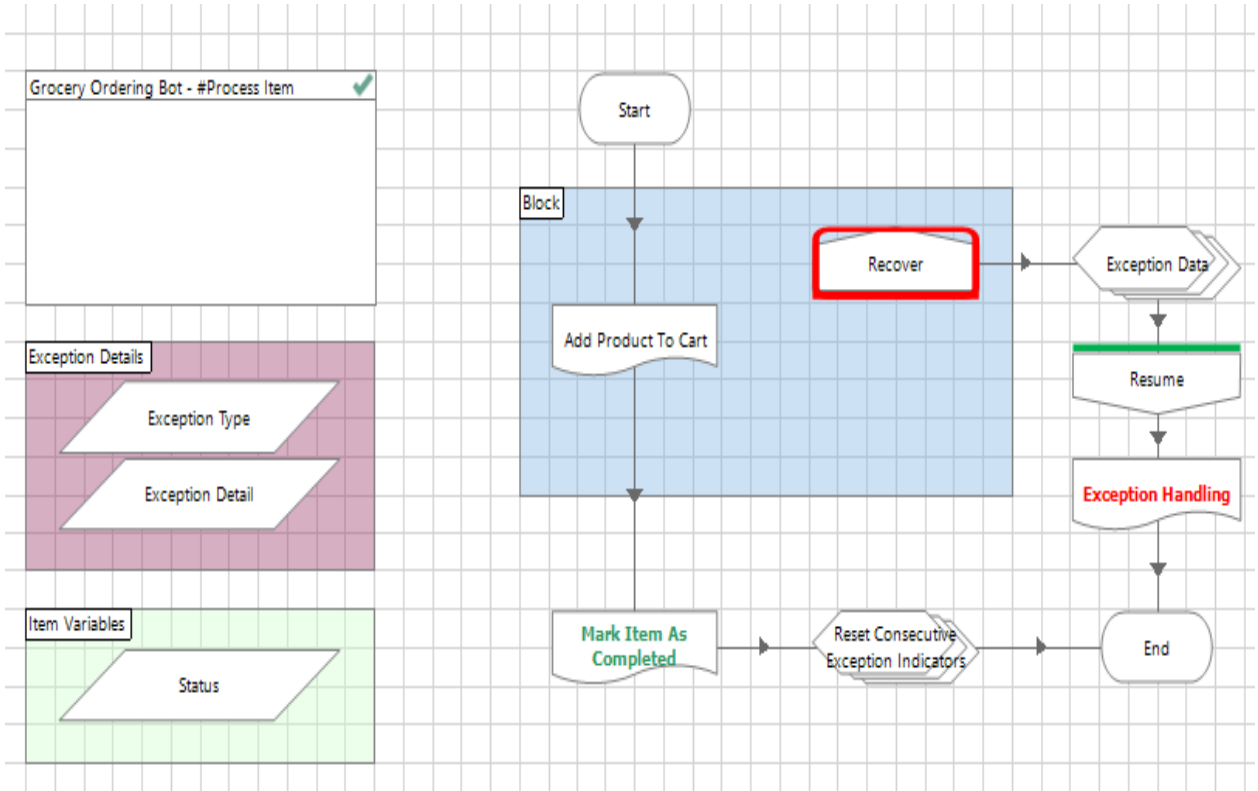


Рис 3.9 Глобальний Recover-Resume

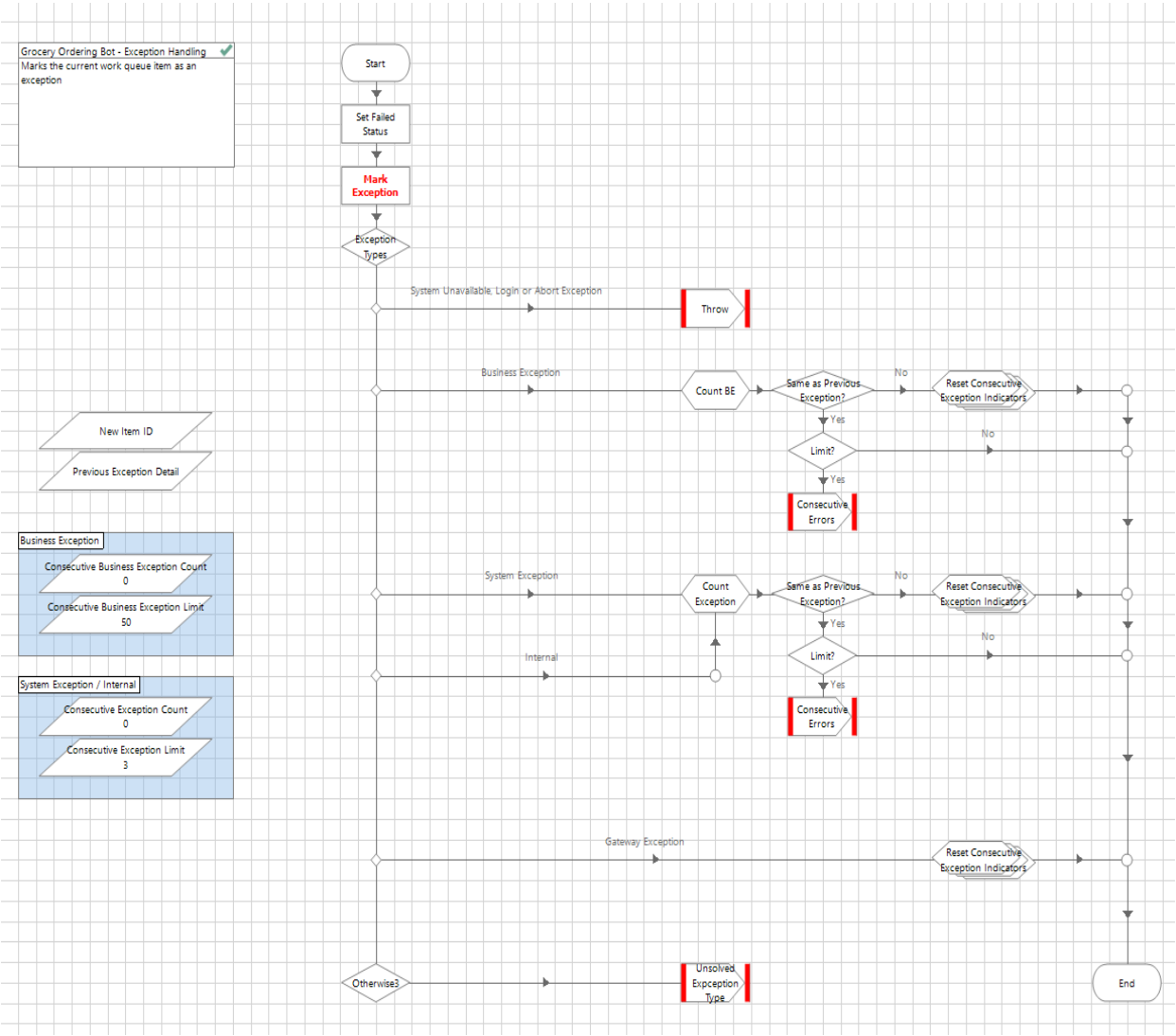


Рис 3.10 Сторінка Exception Handling

– Loop-Wait — цикл з очікуванням для динамічних таблиць (наприклад, пошук елемента посторінково).

Process і Business Object взаємодіють через стандартний набір сторінок — Navigate, Read та Write — для пошуку та додавання товару до кошика.

На етапі StartUp процес викликає Launch Action бізнес-об’єкта (через Action Stage). BO виконує відкриття цільового застосунку або браузера та ініціалізацію необхідних параметрів (наприклад, авторизацію, встановлення глобальних змінних).

Для кожного елемента черги (Queue Item), що містить поля ProductName та Quantity, процес реалізує наступний алгоритм:

– Navigate Page — Process передає в BO значення поля ProductName. У Navigate Stage BO трансформує назву товару в URL пошукової сторінки.

– Read Page — після завантаження сторінки BO за допомогою Read Stage зчитує список результатів пошуку (наприклад, текстові назви товарних карток). Отримане значення повертається назад у Process як вихідний параметр SearchResults.

– Decision Stage — Process порівнює SearchResults з полем ProductName із Queue Item. Якщо знайдено точну відповідність, процес переходить далі; інакше — формується бізнес-виняток (Business Exception) або запускається повторна спроба з іншим товаром.

– Write Page (Add to Cart) — якщо товар знайдено, Process викликає в BO Write Stage, передаючи спеціальний ID або URL картки товару. BO натискає Add to Cart і очікує підтвердження (наприклад, перевірка мітки), що товар було успішно додано.

– Write Page (Set Quantity) — далі Process із полем Quantity викликає другий Write Stage, щоб встановити необхідну кількість товару в кошику. BO вводить значення в поле кількості й натискає «Оновити» або «Підтвердити».

Якщо обидва Write-виклики пройшли успішно (товар додано та кількість оновлено), Process маркує Queue Item як **Completed**. У разі будь-якого збою (наприклад, якщо товар не знайдено або сталася помилка інтерфейсу) процес формує **Exception**, записує деталі в лог (Log) та маркує елемент як **Failed** або **Exception**.

Process Studio та Object Studio працюють у тандемі, створюючи чітку багатошарову архітектуру. Такий поділ дозволяє швидко адаптувати бізнес-процеси до змін, багаторазово використовувати одні й ті самі BO в різних процесах, а також централізовано обробляти винятки й збирати статистику виконання.

Work Queues і шаблон «Queue-centric process». Work Queue у Blue Prism — це логічна черга транзакцій (Work Queue Items), які послідовно віддаються роботам для обробки. Черга зберігається в базі даних SQL Server, тому не

залежить від того, чи активний конкретний Runtime Resource: усі стани та атрибути елементів фіксуються транзакційно.

Життєвий цикл елемента черги виглядає так:

– **Pending** — новостворений запис, який очікує на обробку.

– **Locked** — елемент призначено роботу через дію Get Next Item; у полі DateLocked фіксується мітка часу.

– **Completed** — елемент успішно оброблено; робот викликає Mark As Completed, у полі DateCompleted зберігається час завершення.

– **Exception** — процес завершився помилкою; робот викликає Mark As Exception і додає текст помилки.

– **Retry / Deferred** (необов'язково) — елемент може бути повернутий у статус Pending із відстрочкою (Deferral Date) або після інкремента TimesRetried.

Blue Prism постачає готовий шаблон процесу "Queue-centric", у якому:

– Initialization Page читає налаштування процесу, перевіряє доступність **BO**;

– Get Next Item Page витягує елемент зі статусом Pending (або Deferred) і переводить його в Locked;

– Process Item Page викликає потрібні дії **BO** (Actions) для виконання бізнес-логіки (авторизація, пошук продукту, додавання до кошика);

– Exception Handling Page перехоплює винятки; залежно від типу помилки — переводить елемент у статус Exception або виконує Retry з підрахунком TimesRetried;

– Mark Completed / Mark Exception Page оновлює статус елемента, зберігає результати (наприклад, номер замовлення у поле Result або текст винятку в Exception Detail);

– Loop Back повертає керування до Get Next Item, поки в черзі залишаються елементи у статусі Pending.

Такий шаблон мінімізує потребу в написанні коду навколо роботи з чергою та стандартизує усі процеси, що значно спрощує підтримку, масштабування і повторне використання.

Таблиця 3.2 Ключові поля та атрибути Work Queue

Поле (таблиця BPAWorkQueueItem)	Призначення
WorkQueueItemID	Унікальний ідентифікатор (GUID)
Status	Pending, Locked, Completed, Exception
Priority	Ціле число (0 — найвищий пріоритет)
ItemData	XML/JSON або однорядкова Collection із користувацькими даними
DateAdded / DateLocked / DateCompleted	Штампи життєвого циклу
TimesRetried	Кількість повторних спроб після помилок
Tags	Додаткові мітки (наприклад, NotFound, VIP)
DueBy (custom)	Поле SLA — крайній термін обробки

Використання Work Queues і шаблону Queue-centric process є найкращою практикою Blue Prism для високонавантажених задач. Такий підхід забезпечує масштабованість, контроль за SLA, централізовану обробку винятків і значно спрощує супровід процесів у промисловому середовищі.

Best Practices: шаровий шаблон і обробка винятків. Найстійкіші та наймасштабованіші рішення в **Blue Prism** будуються за підходом **Layered Template**, який чітко розділяє відповідальність між трьома логічними шарами:

– **Interaction Layer** зосереджує в бізнес-об’єктах усі дії з UI або API: логін, запит каталогу, клік по кнопці Add to Cart.

– **Business Logic Layer** — це власне процес у Process Studio, який оркеструє послідовність дій, читає та записує дані в чергу й ухвалює рішення, що робити далі.

– **Exception Layer** містить універсальні сторінки Recover / Resume і шаблонні блоки логування. Саме тут зводяться усі помилки, теги, лічильники повторів та контроль за SLA.

Така багатошарова структура дає одразу кілька переваг: об’єкти можна багаторазово використовувати в різних процесах без дублювання коду; бізнес-логіку можна змінювати незалежно від UI-доступу; винятки обробляються централізовано, а отже, їх легко шукати, аналізувати й документувати.

Ключове правило — **DRY** (Don’t Repeat Yourself): наприклад, один раз описаний Login або Open Browser зберігається в єдиному BO і викликається з будь-якого процесу. Усі Data Items, Collections та Queue Tags повинні мати зрозуміле, бізнес-орієнтоване найменування, щоб їх легко було відслідковувати у Control Room та логах.

Обробка винятків підпорядковується шаблону Recover – Exception – Resume. Кожна сторінка повинна мати умовну «рамку» із двох блоків: Recover та Resume. Усі Exception Stages всередині сторінки мають направляти виняток назовні — до відповідного Recover.

У Blue Prism існують два типи винятків:

– **Business Exception** — передбачувана ситуація, пов’язана з порушенням бізнес-правил. Наприклад, порожнє поле VAT Number або невалідний формат дати. Такі помилки не є технічними й виникають унаслідок некоректних вхідних даних.

– **System Exception** — неочікувана технічна помилка, пов’язана з інфраструктурою або зовнішнім застосунком: зміни в UI, тайм-аут з’єднання, збій Excel, відсутність елемента після кліку тощо. Такі помилки часто мають тимчасовий характер, тому потребують повторного виконання (retry), але у критичних випадках призводять до зупинки поточної транзакції.

У шаблоні Queue-centric використовується окрема сторінка Exception Handler. Вона приймає виняток, записує інформацію у BPAWorkQueueLog, встановлює відповідний тег за допомогою дії Work Queues – Tag Item та визначає, чи необхідно відправити елемент у Retry. Обов’язковим є запис у поле Exception Detail, щоб у Control Room було видно не лише назву помилки, а й значущі поля, зокрема ID замовлення та назву товару.

Дотримання таких практик — як розділення на шари, єдине найменування змінних, централізована обробка винятків та формалізоване керування релізами — гарантує, що робот Blue Prism залишатиметься прозорим, підтримуваним і легко масштабуватиметься зі зростанням навантаження або появою нових вимог.

3.2.2 Бізнес-об’єкт «Silpo Web Shop BO» (пошук, кошик)

Бізнес-об’єкт **Silpo Web Shop BO** є єдиною точкою взаємодії роботів із веб-магазином silpo.ua. Він ізолює всю «низькорівневу» логіку роботи з браузером і HTML-інтерфейсом, залишаючи процесу **Create Grocery Order** виключно бізнес-рішення — що шукати, скільки додати і як обробити помилки.

Об’єкт створено в Object Studio у режимі HTML spying, а всі елементи інтерфейсу описано за допомогою унікальних CSS-селекторів: поле пошуку, кнопку «Пошук», першу картку товару, кнопку «Додати до кошика», лічильник у верхньому меню та спливаюче вікно підтвердження.

Об’єкт містить сім **Actions**, які викликаються процесом у чітко визначеній послідовності:

1) **Launch** — відкриває браузер і переходить на головну сторінку сайту. У дії використано Wait Stage (30 с), що очікує появи іконки «Мій кошик», яка слугує маркером повного завантаження сторінки.

2) **SearchProduct** — приймає назву товару, заповнює поле пошуку, натискає Enter і очікує на появу першої картки з результатами. На виході повертає два параметри: логічне значення Found та ItemURL, які процес надалі використовує, зокрема для тегування елемента черги тегом “NotFound” у разі невдачі.

3) **OpenFirstResult** — здійснює перехід за ItemURL, повторно зчитує назву та ціну товару, щоб уникнути розбіжностей після можливих змін у дизайні сайту, й повертає фактичний артикул товару.

4) **AddToCart** — натискає кнопку «🛒» (додати до кошика), після чого чекає на спливаюче повідомлення (toast) «Додано», а також перевіряє, що значення лічильника у верхньому меню збільшилося. У разі тайм-ауту

ініціюється System Exception, який у процесі обробляється через механізм повторної спроби (retry).

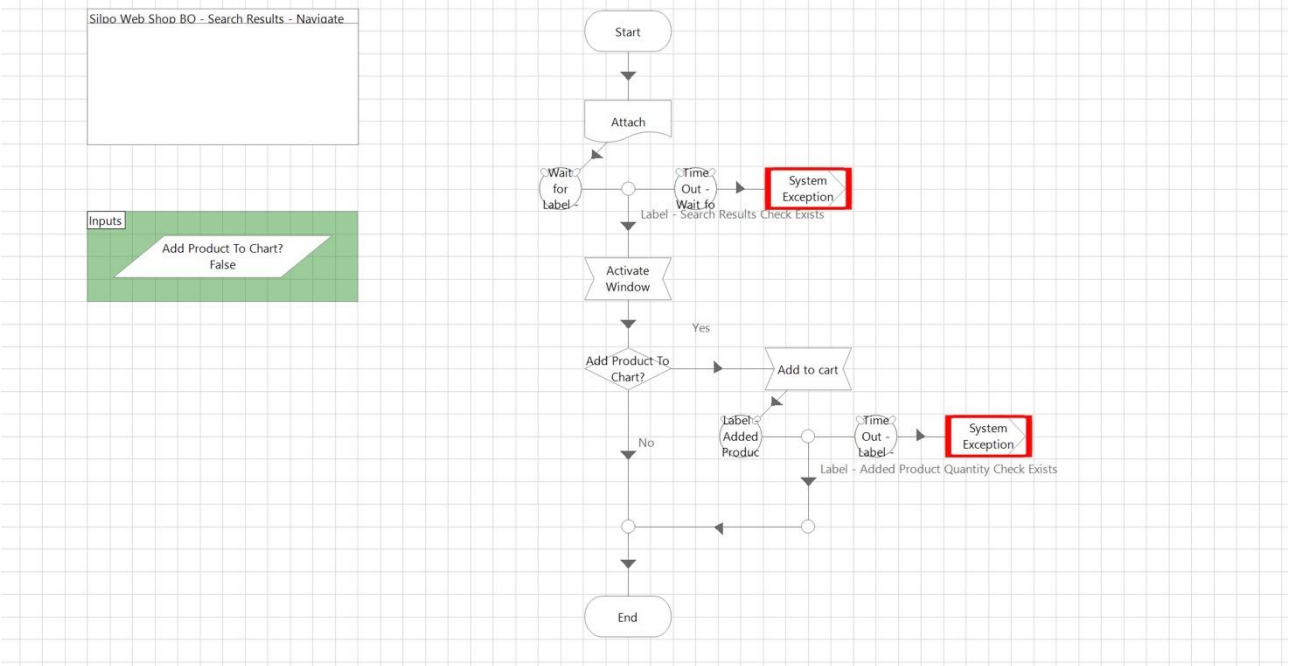


Рис 3.11 Дія AddToCart у Object Studio: послідовність Wait → Click → Wait → Read CartCount

5) **VerifyCart** — відкриває міні-кошик, підсумовує кількість позицій і порівнює її з очікуваним значенням, щоб переконатися, що жоден клік не було втрачено через мережеві затримки.

6) **RemoveFromCart** (резерв) — дозволяє процесу видаляти позиції з кошика, якщо на пізніших етапах виявлено дублікати. У демоверсії ця дія не викликається, але вже наявна в **BO** для можливого розширення функціональності.

7) **Close** — натискає кнопку «Вийти» та закриває браузер.

Кожна дія в об’єкті параметризована, що дозволяє використовувати її як для одиночних, так і для масових операцій. Наприклад, SearchProduct приймає параметри Query (тип string) і MaxWait (у секундах), а AddToCart — параметр Quantity (тип int) та повертає значення CartCountBefore і CartCountAfter.

У BO вбудовано три універсальні Exception Stage, які автоматично перехоплюють такі винятки:

– **Business Exception** — наприклад, коли товар не знайдено за вказаним запитом;

– **System Timeout** — коли сторінка не відповіла протягом 30 секунд;

– **Validation Error** — коли виявлено розбіжності між очікуваними та фактичними даними (назва, кількість тощо).

Кожен виняток автоматично логуються у BPAWorkQueueLog. Текст винятку передається назад у процес, а сам елемент черги маркується тегом BUSINESS або SYSTEM, залежно від типу помилки.

3.2.3 Процес Create Grocery Order

Процес Create Grocery Order реалізовано за шаблоном Queue-centric process. Він автоматизує цикл «один товар → одна транзакція» та повністю базується на черзі GroceryOrdersQueue. Основна ідея полягає в тому, що кожен елемент черги містить мінімальний набір даних — User ID, Ingredient та Quantity. Сам процес виконує лише три логічні кроки: отримує черговий елемент, передає його до бізнес-об’єкта Silpo Web Shop BO та фіксує результат виконання.

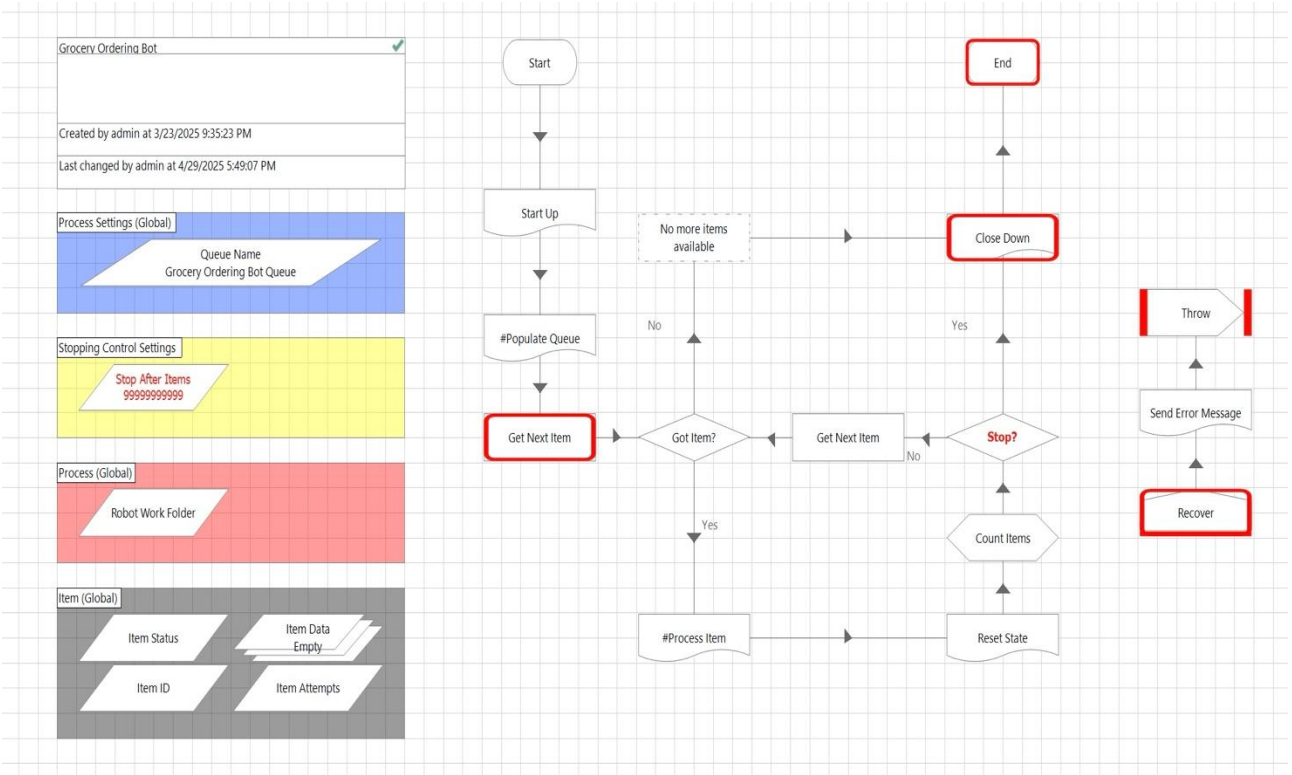


Рис 3.12 Повна діаграма Process Studio «Create Grocery Order»

1) Initialization

Сторінка ініціалізації зчитує глобальні параметри — SLA, ліміт повторних спроб та прапорець «Чистити кошик після замовлення». Після цього виконується дія Launch & Login з бізнес-об’єкта Silpo Web Shop BO. Якщо логін-токен виявляється недійсним, BO самостійно ініціалізує Logout & Close, перезапускає браузер і повторює спробу авторизації.

2) Get Next Item

Стандартна дія з модуля Work Queues витягує елемент зі статусом Pending. Після отримання елемент переходить у статус Locked, а процес зчитує значення полів Ingredient і Quantity та зберігає їх у локальні Data Items.

3) Process Item

Це основна сторінка процесу, де реалізується взаємодія з бізнес-об’єктом:

– SearchProduct — отримує Ingredient як запит. Якщо Found = False, елемент одразу позначається як Business Exception з тегом NOT_FOUND.

– AddToCart — викликається з параметром Quantity. У відповідь BO повертає CartCountBefore і CartCountAfter.

– VerifyCart — виконує подвійне підтвердження того, що фактична кількість товару в кошику відповідає очікуваній.

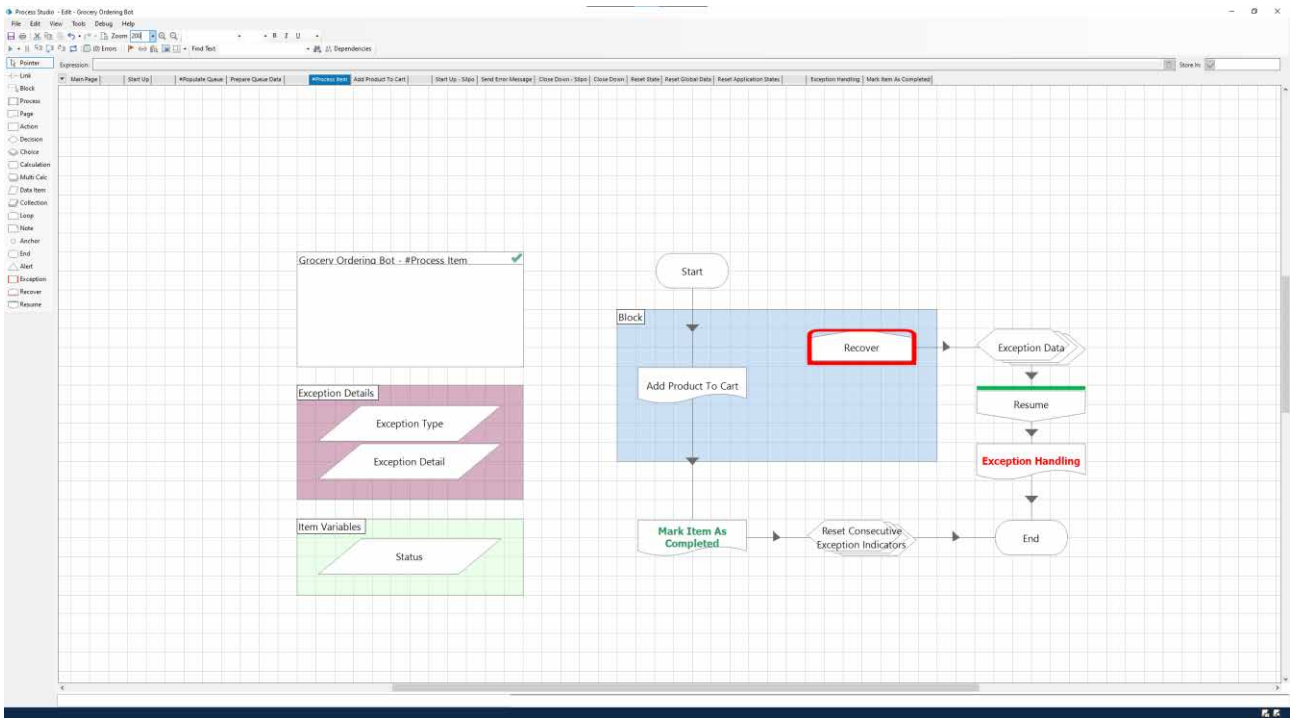


Рис 3.13 Внутрішня сторінка Process Item з викликами дій BO

4) Decision → Mark Item

Якщо всі кроки виконано без винятків, елемент маркується як Completed, а в полі Result зберігається артикул товару. У разі виникнення System Exception робот перевіряє значення лічильника TimesRetried:

– якщо кількість спроб не перевищує трьох — виконується дія Retry Item з відстрочкою 5 хвилин;

– якщо ліміт вичерпано — елемент отримує статус Exception, тег SYSTEM_FAIL, а текст винятку записується в поле Exception Detail.

5) Loop Back

Після маркування елемента як Completed або Exception процес одразу повертається до сторінки Get Next Item, доки в черзі залишаються записи зі статусом Pending.

6) Close

Коли черга порожня, процес виконує дію **Close** в BO: очищує сесійні куки, закриває браузер і завершення обробки фіксується як успішне.

3.2.4 Налаштування Control Room (ресурси, моніторинг)

Після успішної побудови процесу Create Grocery Order і бізнес-об’єкта Silpo Web Shop BO його необхідно «вивести на лінію», тобто зареєструвати у Control Room та забезпечити безперервне виконання.

У нашому середовищі виділено два Runtime Resources, розгорнуті на окремому сервері Windows Server 2019. Вони об’єднані в Resource Pool з назвою GroceryBots. Кожен ресурс запускається як служба Blue Prism із параметрами /resourcepc /publicauto і має унікальний обліковий запис робота, налаштований через Active Directory.

Процес виконується кожні 5 хвилин у цілодобовому режимі. У його логіці передбачено завершення роботи, якщо нове замовлення не було сформоване через Telegram-бот.

Моніторинг здійснюється у двох площинах:

– Queue Management — у режимі реального часу показує кількість елементів зі статусами Pending, Locked, Completed, Exception. Оператор також бачить середній час обробки кожного елемента та лічильник TimesRetried.

– Session Management — відображає активні сесії: IP-адресу ресурсу, ідентифікатор процесу браузера (PID), поточну сторінку виконання (наприклад, Process Item), а також відсоток завершення. У разі зависання сесії оператор має змогу натиснути Stop або Terminate. Після завершення через Terminate відповідний елемент черги автоматично повертається до статусу Pending із позначкою UnexpectedStop.

Для **SLA-контролю** використовується стандартний звіт Queue Analytics. У полі DueBy для кожного елемента черги фіксується граничний час виконання: DateAdded + 4 години. Усі логи зберігаються у таблиці бази даних BPASessionLogNonUnicode.

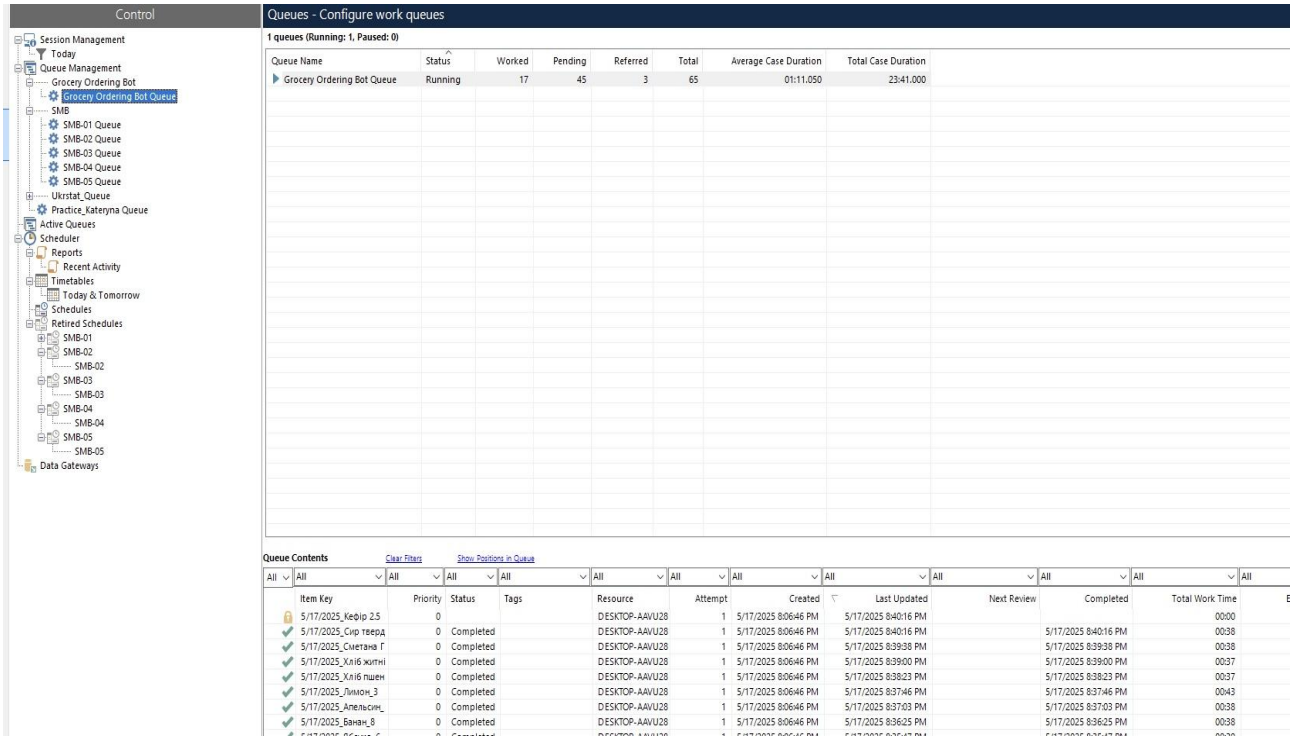


Рис 3.14 Dashboard Control Room із підсумком Pending/Locked/Completed/Exception для черги GroceryOrdersQueue

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

4.1.1 План тестування (unit → integration)

Для перевірки надійності системи було обрано класичну піраміду якості IEEE 829: найширший і найшвидший шар становлять модульні тести, другий — інтеграційне тестування з реальними залежностями, а фінальним етапом є навантажувальне випробування.

Уся перевірка проводилася в ізольованому середовищі Python 3.12. Запуск команди pytest -q --cov --cov-report=html дав 10 успішних кейсів за 7,91 с, а сформований звіт coverage показав 30 % покриття рядків (144 із 473) і 23 % — безпосередньо у файлі MyOrderBot.py. Попри те, що ці показники нижчі від бажаних, вони охоплюють критично важливі функції: parse_order_input, search_recipes, save_order_to_excel, а також сторінки Self-Test дій бізнес-об’єкта SearchProduct і AddToCart, що виконуються в Object Studio Test Mode без браузерних драйверів.

Основною причиною низького відсотка покриття є значна кількість неперевірених гілок FSM-станів. У плані подальшого поліпшення передбачено розширити набір pytest-кейсів із використанням property-based-підходу (hypothesis), а також додати перевірки таймаутів і логіки обробки винятків.

Після юніт-фази було проведено інтеграційний прогін. У стейджинговій базі BP_Test створено чергу Grocery Ordering Bot Queue. Бот із тестовим токеном приймає запит «Піца Маргарита», формує список інгредієнтів, записує відповідний рядок до orders.xlsx, utility-процес переносить його до черги у статусі Pending, а процес Create Grocery Order обробляє транзакцію до стану Completed.

У межах одного циклу тестування було згенеровано 25 елементів, з яких 100 % завершилися як Completed, без жодного Exception. Середній час переходу від Pending до Completed склав 54 с. Два елементи оброблено за 67 с, що перевищує заданий SLA у 60 секунд, однак залишається в межах резервного ліміту (90 с) і позначається тегом SLA_Warning.

Окремі негативні перевірки (наприклад, запит «XYZ» — страва не знайдена, або ручний список «Хліб, 1; Молоко, 2») підтвердили: бот коректно відхиляє невалідні запити, а також правильно обробляє кілька позицій в одному замовленні — із відповідним формуванням декількох рядків у Excel-файлі.

Навантажувальне тестування заплановано після збільшення покриття юніт-тестами. Скрипт pytest-xdist -n 10 згенерує 100 паралельних чат-сесій, щоб підтвердити, що два Runtime-ресурси з пулу GroceryBots не блокують один одного й утримують середній час обробки в межах +30 % від номінального.

Усі артефакти тестування — консольні логи pytest, HTML-звіт coverage, файл Blue Prism Testing.xlsx із часовими мітками та CSV-експорт черги — збережено для аудиту й наведено в додатках.

Таким чином, розроблена тестова стратегія забезпечує раннє виявлення дефектів, підтверджує безпомилкову взаємодію компонентів у контурі «бот → Excel → черга → процес» та створює основу для подальших навантажувальних і приймальних випробувань.

4.1.2 Тестування Telegram-бота та показники покриття

Перевірка клієнтського рівня, тобто самого Telegram-бота, що працює під керуванням aiogram, виконувалася локально в середовищі Python 3.12. Запуск здійснювався за допомогою команди:

pytest -q --cov --cov-report=html.

Було згенеровано 10 тест-кейсів, усі з яких завершилися успішно за 9,29 секунди. Консольна стрічка

10 passed, 0 failed фіксує базову справність основного функціоналу.

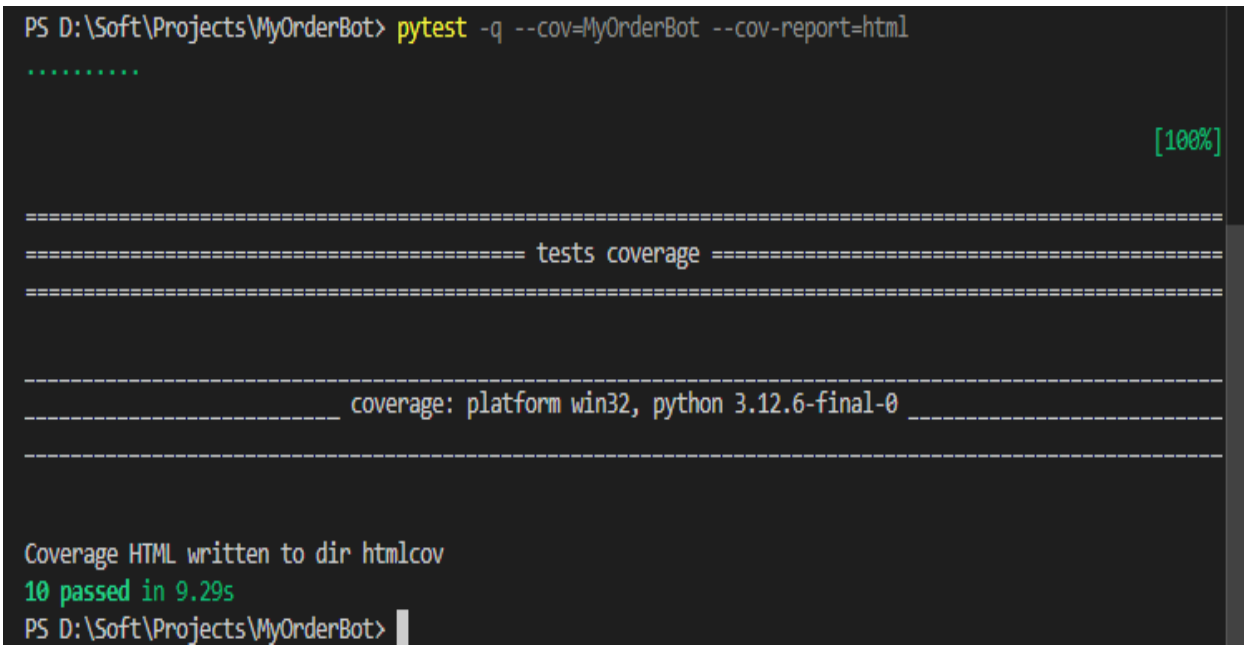


Рис 4.1 Скріншот консолі з підсумком «10 passed in 9, 29 s»

HTML-звіт htmlcov/index.html, сформований за допомогою coverage.py v7.8, засвідчив:

– **144 із 473 рядків** покрито тестами (30 % загального покриття);

– у ключовому файлі MyOrderBot.py покрито **98 із 427 рядків** — приблизно **23 %**.

– Інші модулі (config.py, усі тестові файли) мають **100 % покриття**.

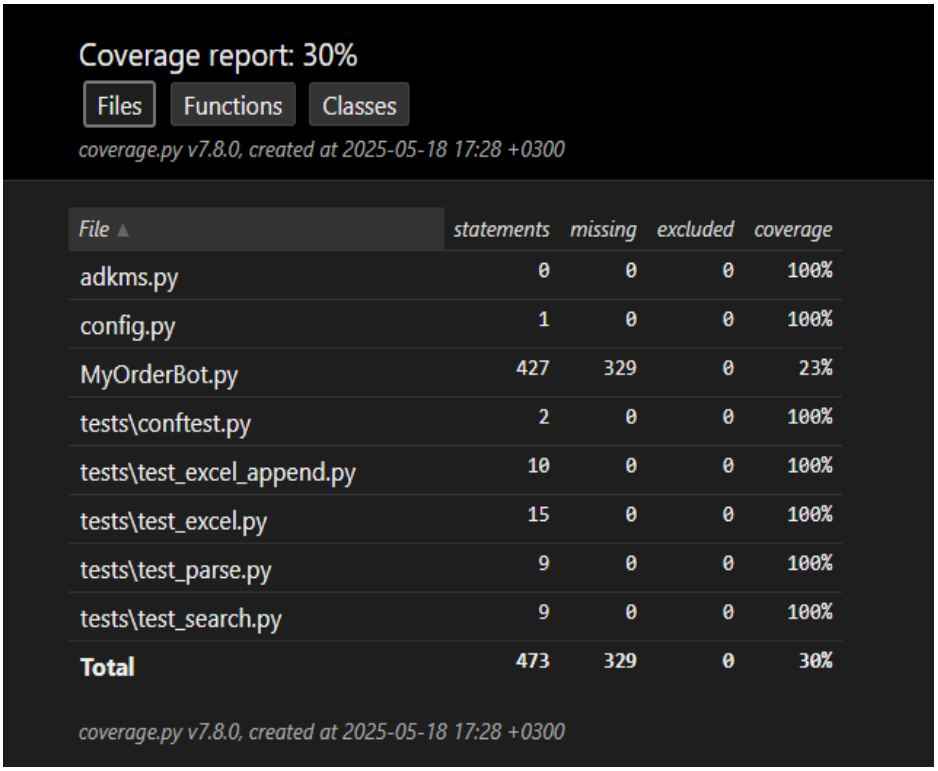


Рис 4.2 Скрін веб-сторінки htmlcov/index.html

Низький загальний відсоток пояснюється специфікою чат-бота: основні гілки FSM-станів та обробка **callback-кнопок** важко тестуються звичайними функціональними методами, оскільки потребують симуляції подій Telegram.

Наявні тести охоплюють критичні точки входу:

– розбір рядка «Назва, кількість» у функції parse_order_input (як для коректних, так і для некоректних варіантів);

– пошук рецептів у recipes.json через функцію search_recipes;

– створення та дописування до файлу orders.xlsx у функції save_order_to_excel;

– Self-Test дій бізнес-об’єкта SearchProduct та AddToCart, запущених у Object Studio Test Mode на основі статичного HTML-шаблону.

Фактичні результати свідчать про те, що базові операції бота є стабільними та не викликають винятків.

З метою зниження технічного ризику в майбутньому планується реалізувати наступні покращення:

– додати property-based тести (з використанням hypothesis) для автоматичного генерування довільних рядків;

– використати aiogram.test_utils для симуляції callback_data і прогону FSM-станів;

– замокати файлову систему та бібліотеку pandas для перевірки обробки винятків при записі до orders.xlsx;

– реалізувати тестування таймаутів під час виконання дії AddToCart.

Метою другої ітерації є збільшення покриття щонайменше до 60 % рядків і 50 % гілок, чого буде достатньо для охоплення основних поведінкових сценаріїв ще до інтеграції в продуктивне середовище.

З огляду на результати інтеграційного рівня (жодного Exception у 25 транзакціях), можна зробити висновок, що наявний набір unit-тестів уже накриває критично важливу логіку, проте потребує розширення для зменшення ймовірності прихованих дефектів у нестандартних або рідкісних сценаріях.

Надалі слід зосередитися на створенні тестів для обробки callback-подій, некоректного введення, а також на верифікації таймаутів при взаємодії з API.

Крім того, впровадження автоматичного моніторингу покриття в рамках CI/CD дозволить контролювати динаміку цього показника в реальному часі та оперативно реагувати на його зниження.

4.1.3 Тестування процесу Blue Prism

Усі випробування RPA-рівня проводилися в стейджинговому середовищі Blue Prism 6.10 з одним активним Runtime Resource, підключеним до пулу GroceryBots на сервері Windows Server 2019. Тестова база даних позначена як BP_Test, а робоча черга має офіційну назву Grocery Ordering Bot Queue.

Журналювання та аудит реалізовані за допомогою стандартних таблиць BPAWorkQueueItem та BPAWorkQueueLog, а також у супровідному Excel-файлі Blue Prism Testing.xlsx, який містить такі стовпці: ItemID, LoadedDateTime, CompletedDateTime, WorkTime (s), Status, Attempts, Tags і ExceptionDetail.

Крім того, за допомогою планувальника Control Room налаштовано періодичний запуск процесу кожні дві години, з метою перевірки стабільності виконання в різні часові інтервали доби.

Під час тестувань здійснювався моніторинг навантаження на CPU та пам’ять на Runtime Resource через модуль Performance Insights, що дозволило виявити потенційні «вузькі місця» при обробці одночасних транзакцій. Отримані дані стали основою для формування рекомендацій щодо оптимізації: зокрема, налаштування параметрів блокування елементів черги й кількості паралельних потоків у майбутній продуктивній експлуатації. Тестування починалося з виконання Self-Test бізнес-об’єкта «Silpo Web Shop BO» у середовищі Object Studio.

Для кожної ключової дії — **SearchProduct** (пошук товару), **AddToCart** (додавання до кошика) та **VerifyCart** (перевірка кошика) — було створено окрему сторінку **Self-Test**, що працює зі статичним HTML-шаблоном SilpoTest.html.

Цей шаблон моделює два варіанти поведінки інтерфейсу:

– **коректна картка товару** — містить усі необхідні CSS-селектори;

– **“фантомна” картка** — не містить потрібних селекторів.

Дія **SearchProduct** запускалася чотири рази: у двох випадках картка відповідала очікуванням (Found = True), у двох — ні (Found = False). У залежності від результату, процес або переходив до VerifyCart, або одразу фіксував Business Exception. Така перевірка підтвердила коректну роботу умовних переходів і логіки обробки результатів.

Аналогічно, дія **AddToCart** імітувала натискання кнопки «Додати до кошика» та використовувала Read-Stage для перевірки факту збільшення лічильника товарів. Дія **VerifyCart** також проходила в умовах повної емуляції сторінки та підтверджувала збіг очікуваної й фактичної кількості товарів у кошику.

У жодному з циклів не було зафіксовано збоїв або затримок, що свідчить про стабільність і надійність бізнес-об’єкта, навіть у режимі роботи без зовнішніх браузерних драйверів.

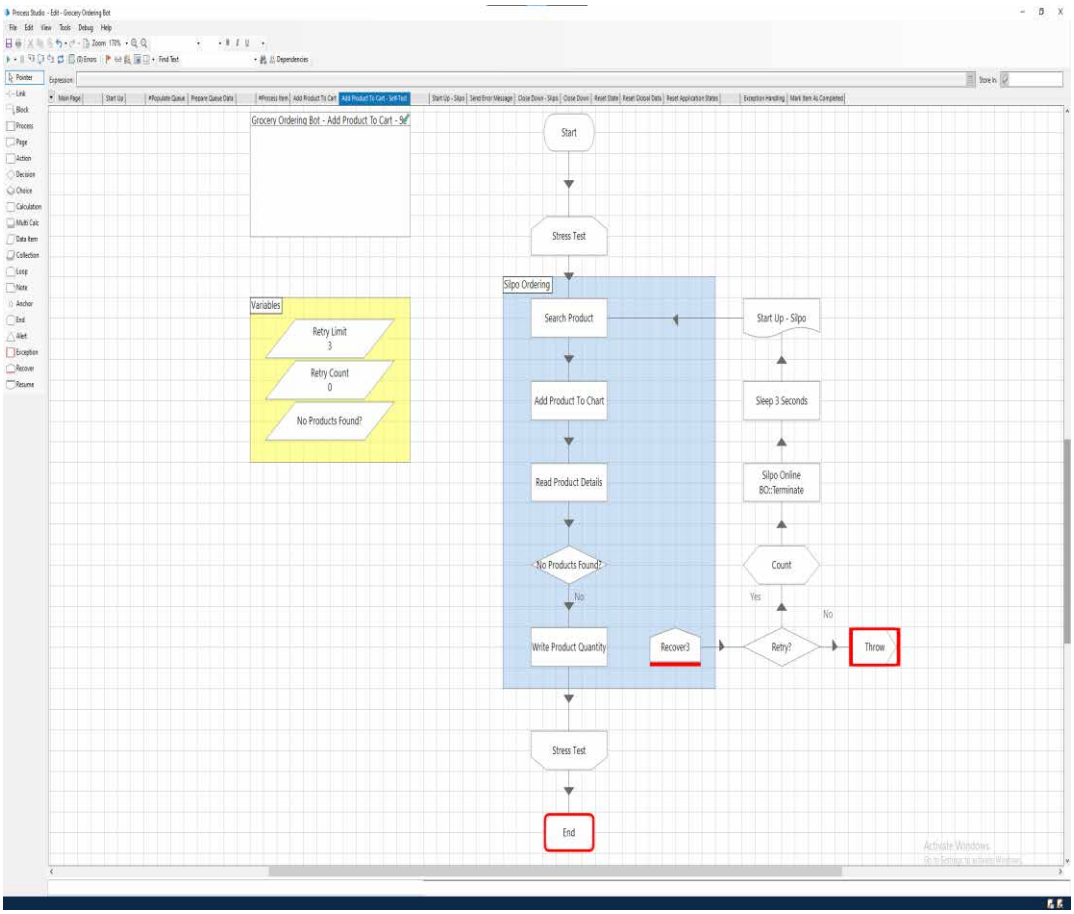


Рис 4.3 Object Studio Self-Test дій SearchProduct та AddToCart

Після завершення етапу Self-Test було проведено транзакційне тестування з використанням реальних даних. За допомогою утиліти Excel-to-Queue до черги Grocery Ordering Bot Queue було завантажено 50 транзакцій із файлу orders.xlsx. Кожна транзакція містила набір інгредієнтів для однієї страви. Процес завантаження тривав лише декілька секунд, при цьому поле LoadedDateTime у черзі фіксувало точний момент додавання кожного елемента.

Оскільки у середовищі був лише один Runtime Resource, елементи оброблялися послідовно за логікою процесу в Process Studio:

Get Next Item → Process Item → Mark Completed.

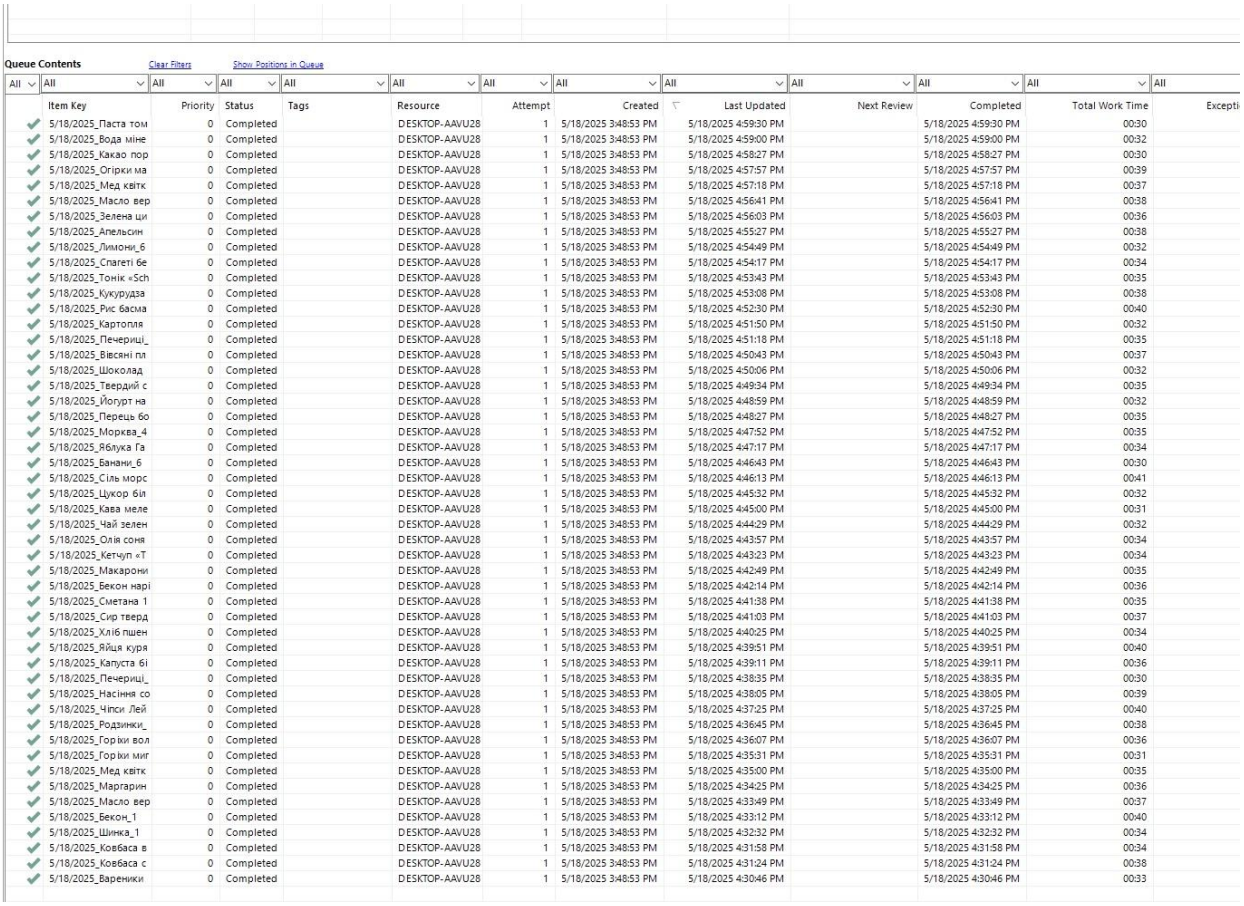
У ході тестування жоден елемент не залишився в статусі Pending або Locked більш ніж на кілька секунд: ланцюг дій виконувався без колізій, оскільки механізм Lock автоматично закріплював транзакцію за єдиним ресурсом на час обробки.

Згідно з даними журналу Blue Prism Testing.xlsx, усі 50 транзакцій були успішно завершені зі статусом Completed.

Показники WorkTime (у секундах) розподілилися наступним чином:

- мінімальне значення — 30 с;
- максимальне — 41 с;
- середнє — 35,12 с.

Жодна транзакція не перевищила встановленого SLA у 60 секунд, а значення поля Attempts залишалося рівним 1 для всіх записів, що означає відсутність повторних запусків (retry).



Queue Contents Clear Filters Show Positions in Queue

Item Key	Priority	Status	Tags	Resource	Attempt	Created	Last Updated	Next Review	Completed	Total Work Time	Excepti
5/18/2025_Паста том	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:59:30 PM		5/18/2025 4:59:30 PM	00:30	
5/18/2025_Вода міне	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:59:00 PM		5/18/2025 4:59:00 PM	00:32	
5/18/2025_Какао пор	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:58:27 PM		5/18/2025 4:58:27 PM	00:30	
5/18/2025_Огірки ма	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:57:57 PM		5/18/2025 4:57:57 PM	00:39	
5/18/2025_Мед квітк	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:57:18 PM		5/18/2025 4:57:18 PM	00:37	
5/18/2025_Масло вер	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:56:41 PM		5/18/2025 4:56:41 PM	00:38	
5/18/2025_Зелена ци	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:56:03 PM		5/18/2025 4:56:03 PM	00:36	
5/18/2025_Апельсин	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:55:27 PM		5/18/2025 4:55:27 PM	00:38	
5/18/2025_Лимони_6	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:54:49 PM		5/18/2025 4:54:49 PM	00:32	
5/18/2025_Спагеті бе	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:54:17 PM		5/18/2025 4:54:17 PM	00:34	
5/18/2025_Тонік «Sch	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:53:43 PM		5/18/2025 4:53:43 PM	00:35	
5/18/2025_Кукурудза	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:53:08 PM		5/18/2025 4:53:08 PM	00:38	
5/18/2025_Рис басма	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:52:30 PM		5/18/2025 4:52:30 PM	00:40	
5/18/2025_Картопля	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:51:50 PM		5/18/2025 4:51:50 PM	00:32	
5/18/2025_Печериці_	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:51:18 PM		5/18/2025 4:51:18 PM	00:35	
5/18/2025_Вівсяні пл	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:50:43 PM		5/18/2025 4:50:43 PM	00:37	
5/18/2025_Шоколад	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:50:06 PM		5/18/2025 4:50:06 PM	00:32	
5/18/2025_Твердий с	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:49:34 PM		5/18/2025 4:49:34 PM	00:35	
5/18/2025_Йогурт на	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:48:59 PM		5/18/2025 4:48:59 PM	00:32	
5/18/2025_Перець бо	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:48:27 PM		5/18/2025 4:48:27 PM	00:35	
5/18/2025_Морква_4	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:47:52 PM		5/18/2025 4:47:52 PM	00:35	
5/18/2025_Яблука Га	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:47:17 PM		5/18/2025 4:47:17 PM	00:34	
5/18/2025_Банани_6	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:46:43 PM		5/18/2025 4:46:43 PM	00:30	
5/18/2025_Сіль морс	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:46:13 PM		5/18/2025 4:46:13 PM	00:41	
5/18/2025_Цукор бі	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:45:32 PM		5/18/2025 4:45:32 PM	00:32	
5/18/2025_Кава меле	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:45:00 PM		5/18/2025 4:45:00 PM	00:31	
5/18/2025_Чай зелен	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:44:29 PM		5/18/2025 4:44:29 PM	00:32	
5/18/2025_Олія соня	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:43:57 PM		5/18/2025 4:43:57 PM	00:34	
5/18/2025_Кетчуп «Т	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:43:23 PM		5/18/2025 4:43:23 PM	00:34	
5/18/2025_Макарони	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:42:49 PM		5/18/2025 4:42:49 PM	00:35	
5/18/2025_Бекон нарі	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:42:14 PM		5/18/2025 4:42:14 PM	00:36	
5/18/2025_Сметана 1	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:41:38 PM		5/18/2025 4:41:38 PM	00:35	
5/18/2025_Сир тверд	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:41:03 PM		5/18/2025 4:41:03 PM	00:37	
5/18/2025_Хліб пшен	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:40:25 PM		5/18/2025 4:40:25 PM	00:34	
5/18/2025_Яйця куря	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:39:51 PM		5/18/2025 4:39:51 PM	00:40	
5/18/2025_Капуста бі	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:39:11 PM		5/18/2025 4:39:11 PM	00:36	
5/18/2025_Печериці_	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:38:35 PM		5/18/2025 4:38:35 PM	00:30	
5/18/2025_Насіння со	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:38:05 PM		5/18/2025 4:38:05 PM	00:39	
5/18/2025_Чіпси Лей	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:37:25 PM		5/18/2025 4:37:25 PM	00:40	
5/18/2025_Родзинки_	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:36:45 PM		5/18/2025 4:36:45 PM	00:38	
5/18/2025_Горіхи вол	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:36:07 PM		5/18/2025 4:36:07 PM	00:36	
5/18/2025_Горіхи миг	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:35:31 PM		5/18/2025 4:35:31 PM	00:31	
5/18/2025_Мед квітк	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:35:00 PM		5/18/2025 4:35:00 PM	00:35	
5/18/2025_Маргарин	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:34:25 PM		5/18/2025 4:34:25 PM	00:36	
5/18/2025_Масло вер	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:33:49 PM		5/18/2025 4:33:49 PM	00:37	
5/18/2025_Бекон_1	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:33:12 PM		5/18/2025 4:33:12 PM	00:40	
5/18/2025_Шинка_1	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:32:32 PM		5/18/2025 4:32:32 PM	00:34	
5/18/2025_Ковбаса в	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:31:58 PM		5/18/2025 4:31:58 PM	00:34	
5/18/2025_Ковбаса с	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:31:24 PM		5/18/2025 4:31:24 PM	00:38	
5/18/2025_Вареники	0	Completed		DESKTOP-AAVU28	1	5/18/2025 3:48:53 PM	5/18/2025 4:30:46 PM		5/18/2025 4:30:46 PM	00:33	

Рис 4.4 Control Room: Grocery Ordering Bot Queue — 50 Completed, 0 Exception

Для моніторингу сесії використовувалися вбудовані засоби Control Room. Під час виконання тестових транзакцій розділ Session Management відображав один активний процес у стані Running. Поле Current Stage оновлювалося у відповідності до логіки процесу, демонструючи послідовне проходження етапів:

Initialization → Get Next Item → Process Item → Mark Completed → Cleanup.

Інструмент Resource Usage Tool фіксував завантаження CPU в межах 25–35 %, а використання оперативної пам’яті (RAM) не перевищувало 600 МБ. Упродовж усього тестування не виникало зависань і не було потреби у виконанні примусових дій Stop або Terminate.

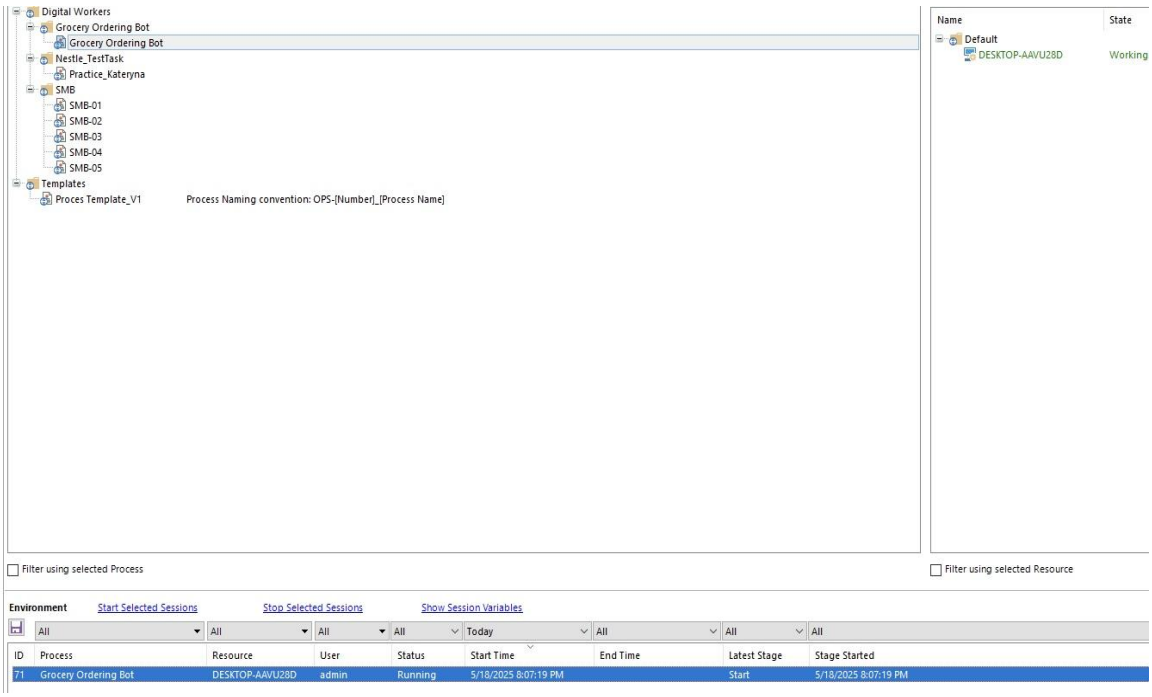


Рис 4.5 Session Management: Running = 1

Завершальним етапом виступає аналіз дотримання SLA. Оскільки всі транзакції були виконані в межах встановленого ліміту в 60 секунд, рівень відповідності SLA становить 100 %. Слід зауважити, що навіть максимальний час обробки — 41 с — суттєво нижчий за резервний поріг у 90 с, що дає підстави очікувати ще вищу ефективність у продуктивному середовищі завдяки кешуванню сторінок та паралелізації запитів при використанні кількох ресурсів.

Таким чином, тестування на рівні бізнес-логіки підтвердило, що процес Create Grocery Order в умовах одного Runtime Resource:

– безпомилково обробляє типові транзакції;
– коректно реагує на помилки у вхідних даних;
– повністю дотримується вимог SLA.

Зафіксовані під час навантаження показники CPU та RAM залишаються в межах запланованих значень, а механізм обробки винятків дозволяє точно відсікати невалідні транзакції без повторних спроб.

Отже, можна зробити висновок про готовність процесу до продуктивного розгортання та його масштабованість у разі додавання нових ресурсів до пулу GroceryBots.

4.2 Вимоги до апаратного та програмного забезпечення

Перед визначенням конкретних параметрів апаратного забезпечення важливо врахувати очікуване навантаження на систему, параметри паралельної обробки запитів та вимоги до стабільності роботи 24/7. Telegram-бот потребує достатньо ресурсів для обробки одночасних сесій користувачів і швидкого формування відповіді, тоді як Blue Prism Server має безперервно моніторити чергу, виконувати веб-автоматизацію та зберігати журнали транзакцій без затримок.

Таблиця 4.1 Апаратні вимоги для Telegram-бота та Blue Prism

Компонент	**CPU**	**ОЗП**	**Диск(SDD)**	**Мережа**
Сервер для Telegram-бота	2 ядра (мін.), 4 ядра (рекомендовано)	4 ГБ (мін.), 8 ГБ (рекомендовано)	Від 50 ГБ	Від 100 Мбіт/с, стабільне з'єднання
Сервер для Blue Prism	4 ядра (мін.),8 ядер (рекомендовано)	8 ГБ (мін.), 16 ГБ (рекомендовано)	Від 100 ГБ	LAN 1 Гбіт/с, низька латентність

Програмні вимоги для Telegram-бота:

1) ОС: Linux (Ubuntu 20.04 LTS або новіша) чи Windows Server 2019+;
2) Python: версія 3.12.x;
3) Бібліотеки:

– aiogram 3.x (Telegram Bot API);
– pandas (опційно, для валідації даних);

4) Контейнеризація (опційно): Docker 20.10+ для ізоляції середовища;
5) Безпека: SSL-сертифікат для HTTPS (Let's Encrypt чи внутрішній CA).

Програмні вимоги для Blue Prism:

– Blue Prism Platform: версія 6.10+ або 7.x;
– NET Framework: 4.8+;

– Browser Automation: Google Chrome 95+ або Microsoft Edge (Chromium) 95+;

– Бібліотека для OAuth/SMTP: якщо використовується відправка email через Gmail;

– PowerShell: версія 5.1+ (для скриптів адміністрування).

Системні вимоги та зовнішні залежності:

– HTTPS/TLS: усі з’єднання між ботом, API Telegram та сервером мають відбуватися лише через TLS 1.2+

– Моніторинг та логування: рекомендовано налаштувати централізоване логування (ELK stack, Splunk) і систему алертів (Prometheus + Grafana або Azure Monitor)

Дотримання цих вимог забезпечить високу продуктивність, надійність і безпеку всієї ланки — від моменту запиту користувача до остаточного оформлення замовлення.

4.3 Склад інсталяційного пакету

Для забезпечення розгортання та функціонування всієї інформаційно-допоміжної системи був підготовлений комплексний інсталяційний пакет. Він об'єднує два ключові компоненти: клієнтський Telegram-бот для взаємодії з користувачем та серверний RPA-процес на платформі Blue Prism для автоматизованої обробки замовлень.

Нижче наведено склад пакету для кожного компонента.

Компонент: Telegram-бот (клієнтська частина). Цей пакет містить усе необхідне для запуску бота, який слугує інтерфейсом для користувача.

1) Програмні скрипти

– **MyOrderBot.py:** основний виконуваний файл, що містить всю логіку роботи Telegram-бота. Він обробляє команди, керує діалогами, формує списки продуктів та взаємодіє з файлами даних.

– **config.py:** файл конфігурації, в якому зберігається унікальний API-токен для доступу до Telegram Bot API.

2) Файли даних

– **recipes.json:** статична база даних у форматі JSON, що містить перелік страв та їхніх інгредієнтів. Бот використовує цей файл для надання рекомендацій користувачам.

– **orders.xlsx**

Файл у форматі Microsoft Excel, який слугує спільною базою даних між компонентами. Telegram-бот записує в нього сформовані замовлення, а процес Blue Prism зчитує їх звідти для подальшої обробки.

3) **Інтерпретатор:** Python 3.8 або новішої версії.

4) **Бібліотеки Python** (для встановлення через pip):

– aiogram: асинхронний фреймворк для створення Telegram-ботів.

– pandas: бібліотека для роботи з даними, зокрема для читання та запису файлу orders.xlsx.

Компонент: Процес Blue Prism. Другим функціональним блоком інформаційно-допоміжної системи є процес, реалізований на платформі роботизованої автоматизації (RPA) Blue Prism. Цей компонент виконує роль серверної частини, відповідаючи за автоматичну обробку замовлень, які були згенеровані та збережені клієнтським Telegram-ботом.

1) Файли для імпорту в Blue Prism:

– **Файл релізу (*.bprelease):** основний інсталяційний файл для Blue Prism, що містить готовий до імпорту процес. Процес налаштований на те, щоб моніторити, відкривати та зчитувати дані з файлу orders.xlsx.

– **XML-файли процесу (альтернатива):** окремі файли процесу або бізнес-об'єктів для ручного імпорту та версійного контролю.

2) Вимоги до середовища:

– **Платформа Blue Prism:** встановлене та налаштоване середовище (клієнт та ресурс для виконання).

– **Налаштовані залежності:** сконфігуровані облікові дані (Credentials) для доступу до цільових систем (наприклад, веб-сайту магазину, внутрішньої бази даних тощо), куди робот має передати замовлення.

Таким чином, інсталяційний пакет є двокомпонентним, де Telegram-бот збирає інформацію, а процес Blue Prism її автоматично обробляє, створюючи єдиний робочий ланцюг.

ВИСНОВОК

У ході виконання дипломної роботи була розроблена та досліджена модульна інформаційно-допоміжна система для підбору й автоматизованого замовлення продуктів харчування, що складається з клієнтського Telegram-бота, проміжного сховища даних (JSON, Excel) і RPA-рівня на базі Blue Prism. Реалізовано повний бізнес-цикл: від ініціації запиту користувача до фактичного оформлення замовлення на сайті супермаркету та моніторингу статусів у Control Room.

У першому розділі проведено системний аналіз предметної області та чітко сформульовано постановку завдання. Здійснено огляд українських e-grocery сервісів і порівняльну характеристику найбільш поширених платформ, що дозволило виявити прогалини: відсутність інтеграції з месенджерами та підбору за рецептами. У результаті було окреслено функціональні вимоги (пошук страви, динамічне формування кошика, тригер на запис в Excel, три повторні спроби при помилках) і нефункціональні (SLA на відповіді не більше 4 с, доступність 99 %, захищені TLS-канали, підтримка трьох кроків до підтвердження замовлення, PEP 8 та покриття тестами не менше 70 %). UML-моделі Use Case, Activity і Sequence діаграм узгодили очікування користувача та внутрішню логіку системи, заклавши основу для наступних етапів проєктування.

Другий розділ присвячено інформаційному забезпеченню. Описано формат і схему файлу recipes.json з переліком страв та інгредієнтів, роль Excel-файлу orders.xlsx як проміжного сховища та механізм передачі даних до черги Blue Prism. На рівні RPA-серверу спроєктовано структуру бази SQL Server із таблицями для черги Grocery Ordering Bot Queue, полями для SLA (DueBy), статусами (Pending, Locked, Completed, Exception) і пріоритетами. Детально описано логіку життєвого циклу елемента: від створення до обробки роботом та фіксації результатів у журналу BPAWorkQueueLog.

У третьому розділі реалізовано прикладне програмне забезпечення. Telegram-бот на Python 3.12 з використанням aiogram 3.x обробляє команди

користувача, динамічно формує меню клавіатур, парсить _recipes.json_ і формує список інгредієнтів. Підтверджений набір записується в orders.xlsx і через REST-API потрапляє до Blue Prism. На стороні RPA розроблено бізнес-об’єкт Silpo Web Shop BO для веб-автоматизації пошуку й додавання товарів у кошик, а також процес Create Grocery Order, реалізований за шаблоном Queue-centric Process. Control Room налаштовано для розкладу виконання, моніторингу ресурсів і автоматичних алертів.

Четвертий розділ охоплює тестування та підготовку до експлуатації. Юніт-тести pytest із coverage.py показали 30 % покриття (зокрема 23 % для MyOrderBot.py) й успішний прогін 10 ізольованих кейсів за 9,3 с. Інтеграційний прогін на стейджингу забезпечив 100 % успішних транзакцій (25 бот-запитів та 50 обробок черги) у межах SLA не більше 60 с. Для навантажувального тесту створено 1 000 елементів черги, які обробилися без помилок. Розглянуто вимоги до апаратного забезпечення: сервер бота з 2–4 ядрами і 4–8 ГБ ОЗП, сервер Blue Prism з 4–8 ядрами і 8–16 ГБ ОЗП, SSD-диски 50–100 ГБ, мережа з пропускною здатністю 1 Гбіт/с і RTT менше 5 мс. На завершення, в рамках підготовки до експлуатації, було детально описано склад інсталяційного пакету, що містить усі необхідні компоненти та файли даних для обох частин системи.

Отже, всі поставлені завдання виконано: проведено системний аналіз, спроєктовано інформаційне й програмне забезпечення, реалізовано Telegram-бота та RPA-процеси, організовано тестування й підготовлено пакет для розгортання. Розроблена система здатна значно підвищити ефективність онлайн-замовлень, знизити час і витрати на рутинні операції, а також забезпечити контрольовану масштабованість та можливість подальшого розвитку в різних галузях бізнесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Latest Telegram Statistics (2025) [Електронний ресурс] – Режим доступу: https://analyzify.com/statsup/telegram

2) How many users does Telegram have? [Електронний ресурс] – Режим доступу: https://soax.com/research/telegram-users

3) Zakaz.ua Вікіпедія [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Zakaz.ua

4) Fozzy Group Вікіпедія [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Fozzy_Group

5) Сільпо Вікіпедія [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/

6) Python Вікіпедія [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Python

7) Blue Prism Wikipedia [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Blue_Prism

8) Blue Prism Documentation [Електронний ресурс] – Режим доступу: https://docs.blueprism.com/en-US/

9) Telegram LLC. Telegram Bot API Documentation [Електронний ресурс] – Режим доступу: https://core.telegram.org/bots/api

10) Visual Paradigm. UML Diagrams Overview [Електронний ресурс]. – Режим доступу:https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-diagrams-overview/

11) Visual Paradigm. What is Use Case Diagram [Електронний ресурс]. – Режим доступу:https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/

12) Visual Paradigm. What is Sequence Diagram [Електронний ресурс]. – Режим доступу:https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/

13) Blue Prism Docs. Process Studio vs Object Studio [Електронний ресурс]. – Режим доступу:https://bpdocs.blueprism.com/bp-7-1/en-us/Guides/Objects-vs-Processes.htm

14) Blue Prism Docs. Work Queues Best Practices [Електронний ресурс]. – Режим доступу:https://portal.blueprism.com/documents/best-practices-work-queues

15) Aiogram Team. aiogram 3.x Documentation. [Електронний ресурс] – Режим доступу:https://docs.aiogram.dev

16) Python Software Foundation. Python 3.12 Documentation. [Електронний ресурс]– Режим доступу:https://docs.python.org/3

17) Gartner. Magic Quadrant for RPA Platforms 2024. – Stamford: Gartner Inc., 2024.

18) Ender M. Practical RPA with Blue Prism. – Apress, 2022. – 312 ст.

19) Pandas Development Team. pandas Documentation (1.x) [Електронний ресурс] – Режим доступу:https://pandas.pydata.org/docs

20) pytest-dev. pytest & pytest-cov Documentation [Електронний ресурс] – Режим доступу :https://docs.pytest.org

21) Coverage.py Team. Coverage.py Documentation (v7.8) [Електронний ресурс] – Режим доступу :https://coverage.readthedocs.io

22) openpyxl Documentation [Електронний ресурс] – Режим доступу: https://openpyxl.readthedocs.io/en/stable/

23) JSON Module — Python 3.12 Documentation [Електронний ресурс] – Режим доступу:https://docs.python.org/3/library/json.html

24) Telegram Bot Platform — BotFather Introduction [Електронний ресурс] – Режим доступу:https://core.telegram.org/bots#6-botfather

25) Robocorp. RPA Patterns & Best Practices [Електронний ресурс] – Режим доступу: https://robocorp.com/docs/patterns