

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБЛЕННЯ ВЕБЗАСТОСУНКУ ДЛЯ ІНТЕЛЕКТУАЛЬНОГО
ПІДБОРУ ТА ПОРІВНЯЛЬНОГО АНАЛІЗУ ТУРИСТИЧНОГО
СПОРЯДЖЕННЯ НА ОСНОВІ ТЕХНОЛОГІЇ TELEGRAM MINI APPS
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-22-2

Родигін О.А.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Вечірська І.Д.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики Кобилін О. А.
(підпис) (прізвище, ініціали)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджментуКафедра ІнформатикиРівень вищої освіти перший (бакалаврський)Спеціальність 122 Комп'ютерні науки
(код і повна назва)Тип програми освітньо-професійнаОсвітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

« _____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУздобувачеві Родигіну Олексію Андрійовичу
(прізвище, ім'я, по батькові)1. Тема роботи Розроблення вебзастосунку для інтелектуального підбору та порівняльного аналізу туристичного спорядження на основі технології Telegram Mini Apps

затверджена наказом університету від 26 травня 2026 року № 435Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 25.06.2026 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, документація платформи Telegram Mini Apps, мікрофреймворк Flask, бібліотека об'єктно-реляційного відображення SQLAlchemy

4. Перелік питань, що потрібно опрацювати в роботі _____

1. Аналіз предметної області та наявних рішень для підбору туристичного спорядження.2. Математичне обґрунтування методу багатокритеріального порівняння та моделі експертної системи.3. Реалізація вебзастосунку Telegram Mini App, тестування та аналіз отриманих результатів

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми, постановка задачі, архітектура вебзастосунку, схема бази даних, екранні форми застосунку, результати тестування.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	13.04.2026	
2	Аналіз завдання, підбір літератури	14.04.26-16.04.26	
3	Аналіз літератури з проблеми, що вирішується	17.04.26-19.04.26	
4	Аналіз існуючих методів розпізнавання	20.04.26-22.04.26	
5	Формування кроків вирішення проблеми	23.04.26-05.05.26	
6	Програмна реалізація	26.04.26-20.05.26	
7	Аналіз отриманих результатів	21.05.26-04.06.26	
8	Оформлення пояснювальної записки	05.06.26-09.06.26	
9	Перевірка на нормоконтроль	15.06.26-16.06.26	
10	Перевірка на плагіат	16.06.26-17.06.26	
11	Рецензування	17.06.26-18.06.26	
12	Підготовка презентації та доповіді	18.06.26-20.06.26	
13	Занесення роботи в електронний архів	06.07.26	
14	Попередній захист кваліфікаційної роботи	06.07.26	

Дата видачі завдання 13 квітня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Вечірська І.Д.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 101 с., 7 табл., 13 рис., 3 дод., 38 джерел.

БАГАТОКРИТЕРІАЛЬНЕ ОЦІНЮВАННЯ, ВЕБЗАСТОСУНОК, ЕКСПЕРТНА СИСТЕМА, ІНТЕЛЕКТУАЛЬНИЙ ПІДБІР, ТУРИСТИЧНЕ СПОРЯДЖЕННЯ, FLASK, SQLALCHEMY, TELEGRAM MINI APP.

Об'єктом роботи є процес підбору туристичного спорядження за заданими параметрами походу для формування персоналізованого переліку та порівняльного аналізу моделей.

Метою роботи є розроблення вебзастосунку, що інтегрується у Telegram-клієнт та забезпечує формування персоналізованого переліку туристичного спорядження за параметрами походу, добір товарів за рівнями ціни, а також порівняльний аналіз вибраних моделей за нормалізованими характеристиками з урахуванням ваг, заданих користувачем.

Результати роботи: спроектовано та реалізовано клієнт-серверний вебзастосунок, доступний як Telegram Mini App; побудовано базу даних із тринадцятьма категоріями товарів та таблицею правил експертного модуля; реалізовано модуль багатокритеріального порівняння з можливістю налаштування ваг.

Область застосування: е-комерція туристичного спорядження, інформаційні сервіси для туристичних клубів, освітні матеріали з безпеки в природному середовищі.

EXPERT SYSTEM, FLASK, INTELLIGENT SELECTION, MULTI-CRITERIA DECISION-MAKING, OUTDOOR GEAR, SQLALCHEMY, TELEGRAM MINI APP, WEB APPLICATION.

The object of the work is the process of selecting outdoor gear according to the specified trip parameters in order to build a personalised checklist and to perform a comparative analysis of the selected models.

The goal of the work is to develop a web application that runs inside Telegram and provides a personalised gear checklist for the trip parameters, suggests products by price tier, and enables a comparative analysis of selected models using normalised characteristics with user-controlled weights.

Results: a client-server web application has been designed and implemented as a Telegram Mini App; the database holds thirteen product categories and a rule table of the expert module; a multi-criteria comparator with adjustable weights has been built.

Application area: outdoor-gear e-commerce, information services for hiking clubs, training materials on safety in the natural environment.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів	7
Вступ	9
1 Аналіз предметної області та постановка задачі	11
1.1 Ринок туристичного спорядження та потреби користувача	11
1.2 Аналіз платформи Telegram Mini Apps	12
1.3 Аналіз аналогів і виявлення прогалин.....	14
1.4 Аналіз літературних джерел	16
1.5 Аналіз теоретичних основ багатокритеріального ухвалення рішень	18
1.6 Дослідження експертних систем на основі продукційних правил	20
1.7 Аналіз технологічного стека серверної частини.....	21
1.8 Постановка задачі	22
2 Математичне обґрунтування вибраних методів та архітектура системи	24
2.1 Формалізація задачі багатокритеріального вибору	24
2.2 Властивості методу простої адитивної згортки та обмеження	26
2.3 Модель експертної системи формування переліку спорядження.....	29
2.4 Алгоритм добору товарів для переліку	31
2.5 Архітектура вебзастосунку: Model-View-Controller та Flask	33
2.6 Схема бази даних і доменна модель	35
2.7 Об'єктно-орієнтоване моделювання сервісних класів	39
2.8 Життєвий цикл запиту в системі та потоки даних.....	41
2.9 Безпекова модель і автентифікація	43
3 Реалізація вебзастосунку, тестування та аналіз результатів	46
3.1 Обґрунтування вибору інструментальних засобів.....	46
3.2 Реалізація моделей предметної області	47
3.3 Реалізація експертної системи	49
3.4 Реалізація компонента багатокритеріального порівняння	55
3.5 Реалізація серверного API.....	61

	6
3.6 Реалізація клієнтської частини Telegram Mini App	63
3.7 Адміністративний інтерфейс та супровід каталогу	66
3.9 Початкове наповнення бази даних	68
3.10 Тестування системи.....	71
3.11 Аналіз отриманих результатів	73
3.12 Рекомендації щодо використання вебзастосунку	77
3.13 Перспективи подальшого розвитку	79
Висновки.....	81
Перелік джерел посилання.....	83
Додаток А.....	89
Додаток Б.....	94
Додаток В.....	99

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних

МКПР – багатокритеріальне ухвалення рішень

ОРВ – об’єктно-реляційне відображення

ПЗ – програмне забезпечення

РП – рекомендаційна підсистема

СУБД – система керування базою даних

ІІІ – штучний інтелект

API – Application Programming Interface (програмний інтерфейс застосунку)

CRUD – Create, Read, Update, Delete (операції створення, читання, оновлення, видалення)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

DTO – Data Transfer Object (об’єкт перенесення даних)

HTML – HyperText Markup Language (мова розмітки гіпертексту)

HTTP – HyperText Transfer Protocol (протокол передавання гіпертексту)

HTTPS – HyperText Transfer Protocol Secure (захищений протокол передавання гіпертексту)

JSON – JavaScript Object Notation (формат подання структурованих даних)

JWT – JSON Web Token (токен на основі JSON для автентифікації)

MCDM – Multi-Criteria Decision-Making (багатокритеріальне ухвалення рішень)

MVC – Model-View-Controller (модель-подання-контролер, архітектурний шаблон)

ORM – Object-Relational Mapping (об’єктно-реляційне відображення)

REST – Representational State Transfer (передавання стану подання)

SAW – Simple Additive Weighting (метод простої адитивної згортки)

SDK – Software Development Kit (комплект для розроблення програмного забезпечення)

SPA – Single Page Application (односторінковий вебзастосунок)

SQL – Structured Query Language (структурована мова запитів)

TMA – Telegram Mini App (мініатюрний застосунок усередині Telegram)

UI – User Interface (інтерфейс користувача)

URL – Uniform Resource Locator (адреса ресурсу в мережі)

UX – User Experience (досвід взаємодії користувача із застосунком)

ВСТУП

Розвиток мобільних месенджерів на початку 2020-х років суттєво змінив структуру цифрового споживання: значна частка інтернет-користувачів отримує контент і завершує транзакції, не покидаючи звичних застосунків спілкування. Telegram перейшов від простого месенджера до платформи, що поєднує передавання повідомлень, голосові й відеовиклики, медіапубліцистику й мікросервіси. У 2022 році команда Telegram оголосила про впровадження Telegram Mini Apps – технології, що дозволяє відкривати повноцінні вебзастосунки безпосередньо у вікні чату через WebView [1]. Унаслідок цього з'явилася можливість створювати інтерактивні сервіси, що не вимагають окремої реєстрації, встановлення з магазину застосунків та повторного введення персональних даних, оскільки використовують підтверджену Telegram-ідентифікацію користувача.

Ринок туристичного спорядження є однією з тих сфер, де користувач часто стикається з проблемою інформаційного перенасичення. У відкритих каталогах присутні тисячі моделей наплічників, наметів, спальників, термобілизни, мембранних курток і черевиків; характеристики цих виробів описуються десятками параметрів, причому одні параметри впливають на безпеку (температурний рейтинг спальника, водотривкість мембрани), інші на комфорт (вага, об'єм наплічника), ще інші на ціну. Початківець, який збирається у дводенний похід або готується до сходження на вершину висотою понад дві тисячі метрів, має ухвалити складне комплексне рішення під значним навантаженням невизначеності. Помилка у виборі несе ризик не лише фінансових втрат, а й втрати здоров'я.

Аналіз існуючих рішень засвідчує два полюси. Перший полюс представляють великі магазини спорядження, у яких пошук побудовано на жорстких фільтрах за категоріями та діапазонами значень. Такий підхід ефективний для досвідченого користувача, який знає, що саме шукає. Другий полюс утворюють тематичні портали з оглядами, де журналісти суб'єктивно

ранжують моделі за словесними оцінками. Між цими полюсами залишається ніша автоматизованого підбору на основі параметрів походу та порівняльного аналізу за нормалізованими характеристиками з можливістю керування вагами – функціональності, що поєднує переваги формальної моделі та експертних знань.

Світові тенденції розв’язання поставленої задачі полягають у поєднанні методів багатокритеріального ухвалення рішень і знаннєвих систем. Численні дослідження останнього п’ятиріччя з багатокритеріальної аналітики свідчать, що метод простої адитивної згортки із мінімаксною нормалізацією залишається найпрозорішим для нефахівця, водночас даючи прийнятну якість ранжування для відкритих параметричних задач [2, 3]. Експертні системи на основі продукційних правил успішно застосовуються там, де знання експерта легко формалізувати у вигляді умова-дія, як, наприклад, у виборі шарів одягу залежно від погодних умов походу [4]. Об’єднання цих методів у межах одного вебзастосунку дозволяє покрити дві паралельні потреби нефахівця: спочатку отримати орієнтаційний перелік спорядження, а далі осмислено порівняти кілька конкретних моделей за обраними критеріями.

Актуальність кваліфікаційної роботи обумовлена відсутністю в українському цифровому просторі інтегрованого вебзастосунку для туристичного спорядження, який поєднував би формування переліку та порівняння, працював усередині Telegram і не вимагав від користувача спеціальних знань щодо матеріалів і технологій. Розв’язання цієї задачі належить до сфери комп’ютерних наук та опрацювання даних і потребує сучасних методів проектування клієнт-серверних систем, баз даних, експертних модулів і алгоритмів багатокритеріальної оптимізації.

Напрями використання отриманих результатів: магазини туристичного спорядження як інструмент консультативного супроводу клієнта; туристичні клуби як засіб підготовки до походів; освітні центри з безпеки в природному середовищі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Ринок туристичного спорядження та потреби користувача

Туристичне спорядження належить до продуктів довготривалого використання з вираженою технічною специфікою. Сама категорія охоплює кілька порівняно автономних кластерів: наплічники для багатоденних походів, намети, спальні мішки, килимки, термобілизна, флісові светри, синтетичні утеплювачі, пухові куртки, мембранні куртки і штани, треккінгове взуття, кухонне приладдя, аксесуари. Каталог сервісів-агрегаторів зазвичай нараховує сотні моделей у кожній категорії, причому виробник декларує власні стандарти випробувань, наслідком чого є складність зіставлення характеристик за документацією.

Дослідницькі огляди останніх років, проведені редакцією OutdoorGearLab та The Trek, демонструють, що найбільш стабільне ядро параметрів вибору формують такі величини: маса виробу, ціна, водотривкість мембрани або тканини в гідростатичних міліметрах, температурний рейтинг (для спальників і термобілизни), об'єм (для наплічників, намету), термічний опір (для килимків) [5, 6]. Решта характеристик, як-от тип пряжки, форма каптура, кількість кишень, наявність бічного входу, відіграють роль другорядних. Це збігається з результатами польових випробувань, у яких автори оглядів формально присвоюють вагу кожній характеристиці та ранжують моделі шляхом зваженого підсумовування нормалізованих оцінок [5].

Сама практика використання спорядження накладає обмеження, які не очевидні для початківця. Так, температурний рейтинг спальника-комфарту відрізняється від рейтингу екстремуму у кілька градусів, причому маркетингова комунікація виробника зазвичай зміщена на користь нижчого числа. Водотривкість мембрани в десять тисяч гідростатичних міліметрів може бути недостатньою для дводенної мокрої погоди, тоді як двадцять тисяч

надмірні для прогулянки містом. Рекомендований об'єм наплічника зростає не лише з тривалістю походу, а й зі співвідношенням маси спорядження до маси тіла учасника. Усі ці нюанси формують специфічну потребу: зробити перший крок у виборі спорядження за параметрами походу, а не за параметрами товару.

Подальший аналіз аудиторії свідчить, що користувач сервісів туристичного спорядження поділяється на три сегменти. Перший, найчисленніший, складають нефахівці, які купують один-два предмети на сезон і ухвалюють рішення під впливом порад продавця або огляду в соціальній мережі. Другий сегмент об'єднує досвідчених туристів, які знають конкретні характеристики й порівнюють кілька моделей між собою. Третій сегмент сформований інструкторами та спеціалізованими школами, які беруть на себе роль експерта для нефахівця. Інтегрований сервіс має забезпечити перевагу для першого сегменту, не розчарувавши при цьому два інші: початківцю надати готовий перелік за параметрами походу, досвідченому учаснику – багатокритеріальне порівняння обраних моделей, інструктору – інструмент адміністрування правил формування переліку.

1.2 Аналіз платформи Telegram Mini Apps

Технологія Telegram Mini Apps з'явилася у відкритому доступі у 2022 році й відтоді розглядається як один із найпотужніших розподільних каналів для вебзастосунків споживчого сегмента. За даними виробника, місячна аудиторія Telegram перевищує дев'ятсот мільйонів користувачів, які інтенсивно споживають інтерактивний контент усередині месенджера, не перемикаючись на сторонні застосунки [1]. З погляду розробника, Mini App є звичайним вебзастосунком, який відкривається у вбудованому WebView Telegram-клієнта і отримує контекст користувача через об'єкт Telegram.WebApp.

Архітектурно Mini App складається з трьох рівнів. Перший рівень утворює клієнтська частина – односторінковий вебзастосунок зі стандартним стеком HTML, CSS та JavaScript; допускається використання сучасних бібліотек подання, таких як React, Vue або Svelte. Другий рівень формує бот, з яким Mini App взаємодіє за протоколом Telegram Bot API: бот надсилає посилання на запуск застосунку, отримує сповіщення з callback-кнопок та розсилає повідомлення-нагадування. Третій рівень реалізується власною серверною частиною розробника, що зберігає бізнес-логіку, базу даних і захищені операції. Серверна частина не залежить від конкретної мови програмування й може бути побудована на Node.js, Python або Java [7].

Один із ключових аспектів технології – автентифікація користувача. Telegram передає клієнтській частині рядок initData, що містить ідентифікатор користувача, ім'я, прізвище, мову інтерфейсу та хеш на основі секретного ключа бота. Серверна частина повторно обчислює хеш за HMAC-SHA-256 і перевіряє відповідність. Алгоритм верифікації є відкритим і не потребує третіх сторін, що відповідає принципу мінімізації довіреного периметра. Водночас у документації наголошено, що сама ідентифікація надає лише контекст і не є гарантією власності або відповідності, тому критичні операції мають супроводжуватися додатковими перевітками на стороні сервера, ідемпотентністю та механізмами повторного відтворення [1].

Серед обмежень платформи варто відзначити нетривкість сесії: Telegram-клієнт може зупинити роботу WebView через нестачу пам'яті операційної системи, у разі переходу у фоновий режим або під час оновлення клієнта; стан інтерфейсу необхідно зберігати на сервері або у постійному сховищі, не покладаючись на тимчасові змінні JavaScript. На додаток до цього обмежено набір нативних API: WebView не має доступу до камери та файлової системи в режимі читання, тому сценарії, що потребують подібних ресурсів, мають делегуватися мобільному застосунку. Для предметної області туристичного спорядження ці обмеження не є критичними, оскільки взаємодія користувача зводиться до вибору параметрів, перегляду переліку, порівняння

та оформлення замовлення [1, 7]. Узагальнену схему взаємодії клієнтської частини Telegram Mini App із серверною частиною наведено на рисунку 1.1.

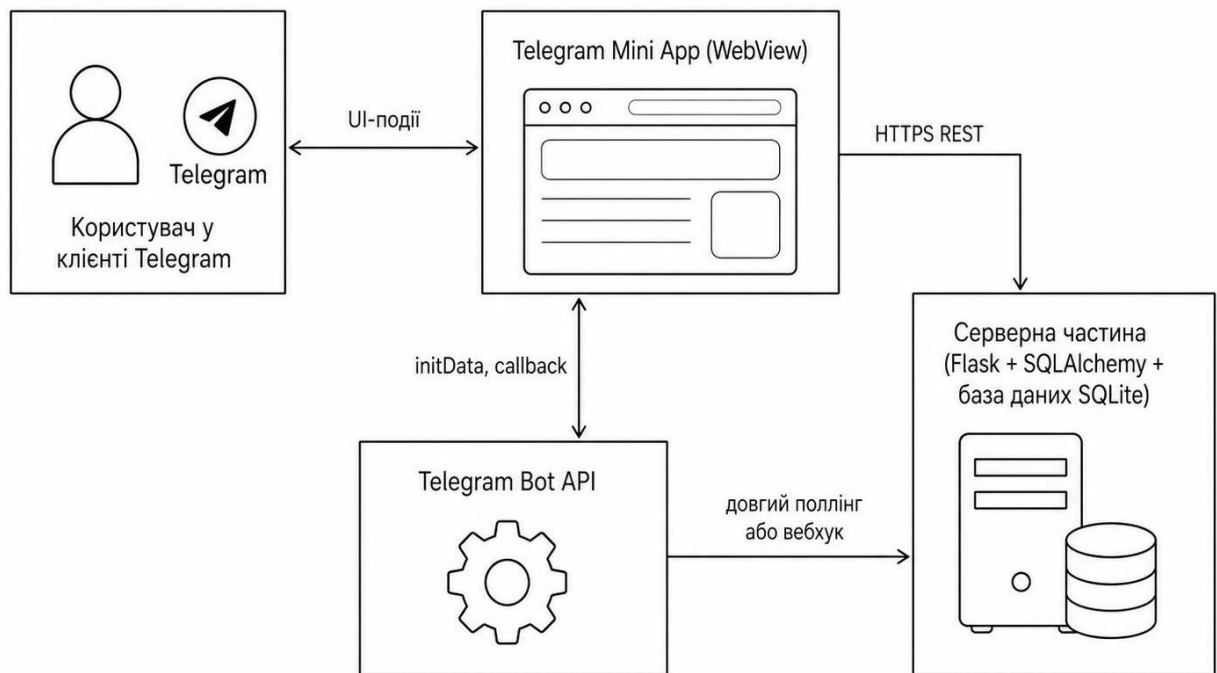


Рисунок 1.1 – Архітектурна схема взаємодії Telegram Mini App, бота та серверної частини

Як випливає з рисунка 1.1, серверна частина одночасно обслуговує два маршрути даних: через REST – безпосередньо з Mini App для відображення інтерфейсу та виконання операцій, через бот – для відправлення повідомлень-нотифікацій та реакції на службові команди. Така розв’язка забезпечує гнучкість і масштабованість, а також відповідає вимогам Telegram до розділення презентаційного та сервісного шарів [7].

1.3 Аналіз аналогів і виявлення прогалин

Для оцінювання стану розв’язання задачі проаналізовано три категорії наявних рішень. Першу категорію утворюють великі магазини туристичного спорядження. Сайти цих компаній містять каталог із багатоступінчастими фільтрами за категорією, призначенням, ціною, масою та маркою. Перевагою

рішень цього класу є повнота асортименту й перевірені дані товарів. Серед недоліків відзначаються відсутність персональної рекомендації за параметрами походу, ручний характер вибору, висока вірогідність помилки початківця, оскільки фільтр не пояснює сенсу обмежень і не співвідносить характеристики виробу з умовами використання.

Другу категорію утворюють спеціалізовані сайти з оглядами та порівняннями (наприклад, OutdoorGearLab, The Trek). Їхні редакційні матеріали містять докладні польові випробування, фотографії, текстові описи й табличні підсумки [5, 6]. Сильною стороною є експертна аналітика, слабкою – статичність списку моделей, що публікується раз на кілька місяців, та прив’язка до зарубіжних брендів і ринків. Користувач не має змоги адаптувати ваги критеріїв до власних потреб або обмежити вибір локально доступними моделями.

Третя категорія охоплює універсальні Telegram-боти-каталоги, що покривають різні товарні групи. Вони пропонують спрощений інтерфейс, але обмежуються кнопками меню та статичними описами без формального обчислення оцінок. У цьому класі бракує саме модуля інтелектуального добору за параметрами походу та модуля порівняльного оцінювання моделей за нормалізованими критеріями. Виявлена прогалина свідчить про доцільність створення спеціалізованого Telegram Mini App, що поєднує переваги великих агрегаторів (повнота каталогу), редакційних оглядів (експертне знання) та ботів-каталогів (доступність усередині месенджера).

Аналіз аналогів узагальнено у вигляді порівняльної таблиці (табл. 1.1), що відображає наявність кожної з ключових функціональних можливостей у трьох категорій рішень.

Таблиця 1.1 – Порівняння функціональності наявних рішень

Функціональна можливість	Магазин	Огляд	Бот
1	2	3	4
Повний каталог товарів	+	–	~

Продовження таблиці 1.1

1	2	3	4
Експертний коментар до моделей	~	+	—
Підбір за параметрами походу	—	—	—
Багатокритеріальне порівняння	—	~	—
Робота всередині Telegram	—	—	+
Налаштування ваг критеріїв користувачем	—	—	—
Адаптація до місцевого ринку	+	~	~

Аналіз даних таблиці 1.1 показує, що жодне з наявних рішень не поєднує одночасно каталогу, експертного підбору, багатокритеріального порівняння та інтегрованої доставки через Telegram. Саме ця комбінація утворює нішу, на яку спрямована кваліфікаційна робота. Джерело даних – власне опрацювання публічних оглядів та документації виробників [1, 5, 6].

1.4 Аналіз літературних джерел

Розв’язувана у кваліфікаційній роботі задача перебуває на перетині трьох напрямів: технологій прийняття рішень в інформаційних системах, формальних моделей задач вибору та інженерії клієнт-серверних вебзастосунків. Підґрунтям для розуміння процесів ухвалення рішень в інформаційних системах слугує навчальний посібник [4], у якому систематизовано моделі подання знань, продукційні правила та механізми логічного виведення; саме ці положення безпосередньо застосовано під час побудови експертного модуля формування переліку спорядження. Методи цифрової обробки зображень, висвітлені у посібнику [8], використано як довідкове джерело під час підготовки графічного вмісту каталогу товарів.

У монографіях [9, 10] та статтях [11, 12] досліджено аналіз багатовимірних даних за описом у формі множини компонент і формування

простору ознак для класифікації. Хоча зазначені праці присвячені аналізу зображень, їхній математичний апарат є методологічно близьким до задачі зіставлення різнорідних характеристик товарів: в обох випадках об'єкт описується вектором неспівмірних ознак, які необхідно звести до спільної шкали. Питання оцінювання значущості ознак, розглянуте у [13], концептуально відповідає проблемі призначення ваг критеріям порівняння, що є центральним елементом розроблюваного модуля. Застосування кластерних та ієрархічних подань даних і гібридних інтелектуальних систем для аналізу динамічних даних продемонстровано у [14–18]; ці результати окреслюють перспективний напрям розвитку рекомендаційного складника застосунку.

Прикладні аспекти створення інтелектуальних сервісів для розв'язання практичних бізнес-задач розкрито у статтях [19, 20]: у першій описано аналіз відеопотоку з камер спостереження для підрахунку відвідувачів, у другій – комбіноване використання великих мовних моделей для генерації тексту. Ці роботи підтверджують доцільність поєднання класичних формальних методів із сучасними інтелектуальними компонентами у складі вебзастосунків.

Особливе значення для кваліфікаційної роботи мають праці, присвячені формальним моделям задач вибору. У статті [21] запропоновано предикатний аналіз даних для побудови системи вибору оптимального місця відпочинку у вигляді логічної мережі; у [22] побудовано математичну модель пошуку танцювальної студії засобами алгебри скінченних предикатів, а у [23] – логічну мережу для розв'язання задачі оренди автомобіля. Формалізацію процесу вибору екскурсії для школярів у формі логічної AFR-мережі описано в [24]. Зазначені роботи демонструють, що задача добору альтернативи за множиною умов природно формалізується системою правил над структурованими фактами; це підтверджує обраний у роботі підхід до побудови продукційної експертної системи переліку спорядження.

Інший клас задач вибору розв'язується методами багатокритеріального аналізу. У [25] побудовано та проаналізовано ієрархічну модель вибору локації для школи, у [26] метод аналізу ієрархій застосовано до задачі вибору

пакету занять у системі здорового харчування, а у [27] – до вибору мовних курсів. Досвід цих досліджень засвідчує практичну придатність зважених згорток критеріїв для споживчих задач вибору і водночас підкреслює вимогу прозорості обчислень для користувача-нефахівця, що враховано під час вибору методу простої адитивної згортки.

Приклад побудови кваліфікаційної роботи з дослідженням методів пошуку зображень у базах даних наведено у [28]. Термінологію у сфері штучного інтелекту в пояснювальній записці вжито відповідно до офіційного словника [29].

Підсумовуючи аналіз літературних джерел, слід зазначити, що формальні моделі задач вибору – продукційні правила, логічні мережі, ієрархічні та зважені згортки критеріїв – неодноразово підтвердили свою продуктивність для споживчих застосувань. Водночас жодне з розглянутих досліджень не поєднує продукційну експертну систему з багатокритеріальним порівнянням у середовищі Telegram Mini Apps, що додатково підтверджує актуальність теми кваліфікаційної роботи.

1.5 Аналіз теоретичних основ багатокритеріального ухвалення рішень

Багатокритеріальне ухвалення рішень об'єднує сукупність методів, що дозволяють упорядкувати або відібрати альтернативи за кількома неспівмірними критеріями. У сучасних оглядах за 2022–2024 роки наголошується, що метод простої адитивної згортки залишається найпрозорішим для нефахівця і часто застосовується як базовий у сервісах онлайн-вибору товарів [2, 3]. Інші підходи – зважений добуток, TOPSIS, АНР, COPRAS – дають близькі за структурою результати, але вимагають додаткових кроків (попарне порівняння критеріїв у АНР, побудова ідеального та антиідеального розв'язку у TOPSIS), що ускладнює інтерпретацію для користувача [30].

Першим обов'язковим етапом будь-якого з названих методів є нормалізація вихідних характеристик. Серед поширених технік нормалізації виокремлюють лінійну з максимумом, мінімаксу, векторну та логарифмічну [3]. Дослідження порівняльної ефективності нормалізаційних схем для різних методів багатокритеріального ухвалення рішень показали, що мінімаксна нормалізація є найбільш робастною для зваженої суми, у якій відбувається компенсація меншої ознаки кращою [3, 31]. Перевагою мінімаксної нормалізації є те, що вона утримує перетворені значення у замкненому діапазоні від нуля до одиниці, що сприяє інтуїтивному розумінню підсумкового бала користувачем-нефахівцем.

Загальна схема багатокритеріального оцінювання у задачі вибору товарів складається з таких етапів. Спочатку формується матриця рішень, рядки якої відповідають альтернативам, а колонки – критеріям. Далі для кожного критерію виконується нормалізація: для критеріїв-переваг (більше – краще) застосовується пряма мінімаксна нормалізація, для критеріїв-витрат (менше – краще) виконується інверсна форма. Потім кожна нормалізована оцінка множиться на нормовану вагу відповідного критерію, а отримані значення підсумовуються в один бал альтернативи. На останньому кроці альтернативи впорядковуються за спадом бала, визначаючи переможця та ранжовану послідовність [2, 3, 32].

Стосовно меж застосовності методу зваженої суми у літературі наголошується, що метод припускає компенсаційну природу критеріїв, тобто допускає можливість компенсувати слабкий показник одного критерію сильним показником іншого. Для туристичного спорядження така компенсаційність прийнятна у більшості випадків (легший намет дорожчий, тепліший спальник важчий), але може бути небажаною для критеріїв безпеки, де нижче порогове значення є абсолютно неприпустимим. Ця обставина враховується у запропонованій моделі шляхом попереднього експертно-фільтрувального етапу, що відсікає моделі, які не відповідають мінімальним вимогам параметрів походу.

1.6 Дослідження експертних систем на основі продукційних правил

Продукційні експертні системи – клас програм, що формалізує знання предметного експерта у вигляді скінченної множини правил «якщо–то». Це найдавніший і досі найпоширеніший формалізм у класичній галузі штучного інтелекту, оскільки правила легко читаються спеціалістом без додаткової підготовки, а саме виведення зводиться до прямого або оберненого ланцюжка міркувань. Сучасні дослідження свідчать про активне застосування продукційних систем у вузькоспеціалізованих сервісах, де знання експерта стабільне, а обсяг правил вимірюється сотнями [4]. Туристичне спорядження потрапляє саме до таких задач: правила формулюються у термінах «активність–сезон–температурний діапазон–тривалість» і відображаються в перелік шарів одягу та груп спорядження.

Класична експертна система складається з чотирьох компонентів: бази правил, бази фактів, інтерпретатора правил і модуля пояснення. У спрощеній реалізації вебзастосунку для туристичного спорядження достатньо перших трьох компонентів. База правил зберігається в таблиці бази даних із полями активність, сезон, температурний мінімум, температурний максимум, мінімальна тривалість, максимальна тривалість, тип шару, назва позиції, опис, пріоритет, рекомендована категорія товарів. База фактів формується клієнтською частиною у момент заповнення форми параметрів походу. Інтерпретатор послідовно перебирає активні правила за зростанням пріоритету, перевіряє кожне на сумісність з фактами та повертає упорядкований перелік позицій спорядження.

Перевагою такої архітектури є не лише прозорість для адміністратора, що супроводжує сервіс, а й відсутність потреби в навчальній вибірці. На відміну від моделей машинного навчання, що потребують збору тренувальних даних, продукційна система працює з першого дня, спираючись винятково на знання експерта. Цей аспект важливий для кваліфікаційної роботи, яка у

першу чергу демонструє інженерну зрілість і здатність формалізувати предметну область, а не повторити готову нейронну мережу.

1.7 Аналіз технологічного стека серверної частини

Серверна частина вебзастосунку реалізується мовою програмування Python з використанням мікрофреймворку Flask. Цей вибір зумовлений кількома міркуваннями. По-перше, Flask надає мінімалістичне ядро, у якому архітектурні рішення приймаються розробником, а не нав'язуються фреймворком; це збігається з вимогою методичних настанов щодо обґрунтованого вибору компонентів. По-друге, наявність широкої екосистеми розширень дозволяє підключити лише потрібні модулі: Flask-SQLAlchemy для об'єктно-реляційного відображення, Flask-Admin для адміністративної панелі, Flask-Limiter для обмеження частоти запитів, Flask-CORS для налаштування міжсайтового обміну [33]. По-третє, Python-стек є природним середовищем для майбутніх удосконалень, у яких передбачається застосування методів машинного навчання для уточнення ваг критеріїв.

Другим компонентом серверного стека є SQLAlchemy – універсальна бібліотека об'єктно-реляційного відображення для Python. Її ключова перевага полягає в розділенні опису предметної моделі та фактичної реляційної бази, що уможливлює перехід з SQLite на PostgreSQL без переписування коду доменних класів [34]. У межах кваліфікаційної роботи на етапі розроблення використовується SQLite, оскільки вона надає вбудовану підтримку у стандартній бібліотеці Python та зручна для тестування на локальній машині. На етапі виробничого розгортання типовим вибором є PostgreSQL, який добре поєднується з асинхронними запитами та гарантує транзакційну цілісність.

Третій компонент – aiogram – асинхронна бібліотека для побудови Telegram-ботів [7]. Через неї реалізовано службові функції, як-от відправлення посилання запуску Mini App, нотифікація менеджера про нове замовлення, обробка callback-сигналів. Завдяки використанню Python-стека обидві частини

серверної інфраструктури мають єдиний середовищний простір, що спрощує розгортання та обслуговування.

1.8 Постановка задачі

Безпосередня покупка туристичного спорядження сьогодні відбувається або у спеціалізованих магазинах, або через універсальні агрегатори, у яких відсутні модулі інтелектуального підбору за параметрами походу та модуль порівняння за нормалізованими характеристиками з налаштуванням ваг. Невідповідність між наявними рішеннями та потребами користувача-початківця обумовлює актуальність кваліфікаційної роботи. Наявність технології Telegram Mini Apps надає природний канал доставки сервісу: користувач звертається до системи всередині звичного для нього месенджера, без додаткової реєстрації та встановлення.

Об'єктом роботи є процес підбору туристичного спорядження за заданими параметрами походу для формування персоналізованого переліку та порівняльного аналізу моделей.

Метою роботи є розроблення вебзастосунку, що інтегрується у Telegram-клієнт та забезпечує формування персоналізованого переліку туристичного спорядження за параметрами походу, добір товарів за рівнями ціни, а також порівняльний аналіз вибраних моделей за нормалізованими характеристиками з урахуванням ваг, заданих користувачем.

Для досягнення мети необхідно вирішити такі завдання:

- проаналізувати предметну область, наявні рішення для рекомендації туристичного спорядження та сформулювати вимоги до функціональності розроблюваного сервісу;

- дослідити можливості платформи Telegram Mini Apps для розгортання вебзастосунку, описати життєвий цикл сесії та схему автентифікації користувача через initData;

- обґрунтувати вибір методу багатокритеріального оцінювання, побудувати математичну модель порівняння товарів та формальну модель експертної системи переліку спорядження;

- спроектувати архітектуру серверної та клієнтської частин, схему бази даних із підтримкою тринадцяти товарних категорій, правил та шкал ваг;

- реалізувати модулі переліку спорядження, інтелектуального добору товарів за рівнями ціни та порівняння моделей за обраними критеріями;

- розробити адміністративний інтерфейс для керування каталогом, правилами експертної системи та шкалами ваг;

- провести модульне й функціональне тестування системи, оцінити ефективність застосунку та визначити напрями подальшого розвитку.

Очікуваним результатом є функціональний вебзастосунок Telegram Mini App, що покриває тринадцять категорій туристичного спорядження, реалізує продукційну експертну систему формування переліку, метод простої адитивної згортки для порівняння товарів та адміністративну панель для супроводу. Практична цінність полягає у скороченні часу та когнітивного навантаження користувача-початківця при підготовці до походу, а також у підвищенні якості рішень за рахунок прозорого формального оцінювання альтернатив. Пояснювальну записку оформлено відповідно до вимог національних стандартів [35, 36].

Особливу складність цьому процесу додає різноманітність високотехнологічних матеріалів. Наприклад, порівняння мембранних тканин, таких як Gore-Tex, у виробках від різних преміальних брендів, як Marmot, Mountain Hardwear чи Karpos, вимагає від користувача глибокого розуміння специфіки паропроникності та зносостійкості. Для непідготовленого туриста такий обсяг технічної інформації часто стає перешкодою при виборі.

2 МАТЕМАТИЧНЕ ОБҐРУНТУВАННЯ ВИБРАНИХ МЕТОДІВ ТА АРХІТЕКТУРА СИСТЕМИ

2.1 Формалізація задачі багатокритеріального вибору

Задача багатокритеріального вибору моделі туристичного спорядження формалізується таким чином. Нехай задано скінченну множину альтернатив A однієї категорії товару (наприклад, наплічники для багатоденних походів) потужності t . Нехай також задано скінченну множину критеріїв C потужності n , що описують релевантні характеристики виробу. Початкова інформація задається у формі матриці X розміру t на n , рядки якої відповідають альтернативам, а колонки – критеріям; елемент x_{ij} має сенс значення j -го критерію для i -ї альтернативи. Кожному критерію присвоюється вага w_j , що відображає його значущість для користувача; ваги задовольняють умовам невід’ємності та одиничної суми.

Перед обчисленням підсумкового бала альтернативи виконують нормалізацію матриці X у матрицю R з елементами r_{ij} , які лежать у діапазоні від нуля до одиниці. Серед нормалізаційних схем, поширених у сучасній літературі з багатокритеріального ухвалення рішень, у роботі обрано мінімаксну нормалізацію, оскільки вона є робастною для методу простої адитивної згортки та інтуїтивно зрозумілою для нефахівця [3]. Для критеріїв-переваг, де більше значення означає кращу якість, нормалізація має такий вигляд:

$$r_{ij} = \frac{x_{ij} - \min_i x_{ij}}{\max_i x_{ij} - \min_i x_{ij}}, \quad (2.1)$$

де r_{ij} – нормалізоване значення j -го критерію для i -ї альтернативи спорядження;

x_{ij} – фактичне значення j -го критерію для i -ї альтернативи;

$\min_i x_{ij}$ – мінімальне фактичне значення j -го критерію серед усіх альтернатив;

$\max_i x_{ij}$ – максимальне фактичне значення j -го критерію серед усіх альтернатив.

Для критеріїв-витрат, де менше значення означає кращу якість (як, наприклад, ціна або маса наплічника), використовується інверсна форма:

$$r_{ij} = \frac{\max_i x_{ij} - x_{ij}}{\max_i x_{ij} - \min_i x_{ij}}. \quad (2.2)$$

У граничному випадку, коли максимальне значення критерію збігається з мінімальним для всіх альтернатив, всі нормалізовані значення дорівнюють одне одному; для уникнення поділу на нуль таким альтернативам присвоюється спільне значення, що дорівнює половині, тобто 0,5. Така евристика збігається з реалізаціями у бібліотеках для багатокритеріальної аналітики й не впливає на ранжування, оскільки розмір кроку нормалізації нульовий.

Метод простої адитивної згортки обчислює підсумковий бал кожної альтернативи як зважену суму нормалізованих значень критеріїв:

$$s_i = \sum_{j=1}^n w_j \cdot r_{ij}, \quad i = 1, 2, \dots, m, \quad (2.3)$$

де s_i – підсумковий інтегральний бал i -ї альтернативи спорядження;

w_j – ваговий коефіцієнт значущості j -го критерію;

r_{ij} – нормалізоване значення j -го критерію для i -ї альтернативи;

n – загальна кількість критеріїв оцінювання;

m – загальна кількість альтернатив у аналізованій категорії товарів.

Альтернативи впорядковуються за спадом значень s_i ; кращою вважається альтернатива з найвищим балом. У сучасних оглядах це рішення

називають найпрозорішим серед методів багатокритеріальної аналітики, оскільки користувач легко простежує внесок кожного критерію у підсумкову оцінку [2]. У межах розроблюваного вебзастосунку метод простої адитивної згортки реалізовано у класі ProductComparator, повний лістинг якого наведено в додатку А.

Особливу увагу слід приділити нормуванню ваг w_j . Користувач, що бажає підкреслити значущість легкої маси наплічника, має змогу збільшити вагу цього критерію через інтерфейс. Програмна реалізація завжди приводить ваги до одиничної суми, що убезпечує від помилок ручного введення:

$$\tilde{w}_j = \frac{w_j}{\sum_{k=1}^n w_k}, \quad (2.4)$$

де $\tilde{w}_j$ – нормоване значення вагового коефіцієнта j -го критерію;

w_j – початкове значення ваги j -го критерію, встановлене користувачем;

w_k – початкове значення ваги k -го критерію;

n – загальна кількість критеріїв оцінювання.

2.2 Властивості методу простої адитивної згортки та обмеження

Метод простої адитивної згортки належить до компенсаційних методів багатокритеріального ухвалення рішень. Це означає, що погіршення значення одного критерію може бути компенсоване покращенням іншого, причому на величину, пропорційну добутку приросту на вагу. Ця властивість прийнятна для більшості сценаріїв вибору туристичного спорядження. Так, легший намет з більшою ймовірністю має нижчу водотривкість, але вигає у вазі компенсує менш суворі вимоги до водотривкості за умови літнього походу. Для зимових походів, навпаки, водотривкість і температурний рейтинг мають бути не нижчими за критичний поріг, оскільки порушення цих вимог несе пряму загрозу здоров'ю.

Для унаочнення роботи методу простої адитивної згортки розглянемо приклад порівняння трьох наплічників однієї категорії за трьома критеріями: ціною (грн, менше – краще), масою (г, менше – краще) та водотривкістю тканини (мм водяного стовпа, більше – краще). Нехай модель А має показники 4500 грн, 1900 г, 8000 мм; модель В – 6200 грн, 1500 г, 12000 мм; модель С – 5400 грн, 1700 г, 10000 мм. Вагові коефіцієнти критеріїв беруться за замовчуванням: 0,35 для ціни, 0,35 для маси та 0,30 для водотривкості.

Після мінімаксної нормалізації за формулами (2.1) і (2.2) отримуємо нормалізовані значення. За критерієм ціни: А – 1,00; В – 0,00; С – 0,47. За критерієм маси: А – 0,00; В – 1,00; С – 0,50. За критерієм водотривкості: А – 0,00; В – 1,00; С – 0,50. Підставивши ці значення у формулу (2.3), дістаємо підсумкові бали: для моделі А – 0,350, для моделі В – 0,650, для моделі С – 0,490. Отже, порядок ранжування має вигляд $V \rightarrow C \rightarrow A$, а переможцем визнається модель В. Приклад наочно демонструє компенсаційну природу методу: попри найвищу ціну, модель В перемагає завдяки виграшу за масою й водотривкістю, які в сумі переважають програш за ціною.

Вибір методу простої адитивної згортки серед інших методів багатокритеріального ухвалення рішень обґрунтовано вимогами предметної області. Метод зваженого добутку, на відміну від адитивної згортки, перемножує нормалізовані значення у степені ваг і різкіше карає низькі показники окремих критеріїв, проте втрачає інтуїтивну прозорість підсумкового бала для нефахівця. Метод TOPSIS, що впорядковує альтернативи за відстанню до ідеального й антиідеального розв'язків, дає якісніше ранжування для великих вибірок, однак потребує обчислення евклідових відстаней і складніше пояснюється кінцевому користувачеві [31]. Метод аналізу ієрархій вимагає попарних порівнянь критеріїв із перевіркою узгодженості суджень, що накладає на пересічного туриста надмірне когнітивне навантаження.

Для розроблюваного вебзастосунку визначальними є три чинники: прозорість підсумкової оцінки, мінімальні вимоги до підготовки користувача

та обчислювальна простота на боці сервера. За всіма трьома чинниками метод простої адитивної згортки переважає згадані альтернативи, тому саме його обрано як основу модуля порівняння. Водночас архітектура модуля ProductComparator не прив'язана жорстко до конкретної згортки: заміна методу обчислення підсумкового бала локалізується в одному методі calculate_score, що залишає простір для переходу на TOPSIS у разі майбутнього масштабування каталогу [32].

Для врахування критичних порогів у запропонованій моделі реалізовано двоступеневу схему. На першому ступені, через експертну систему правил, з усього каталогу відсікаються моделі, що не задовольняють жорстких обмежень за активністю, сезоном і температурним діапазоном. Лише моделі, які пройшли відсічення, передаються до модуля простої адитивної згортки для подальшого ранжування. Така схема поєднує переваги символічної логіки (жорсткі обмеження безпеки) та чисельної аналітики (тонке ранжування за компромісними критеріями), що відповідає сучасним рекомендаціям зі сполучення MCDM з експертними системами [2].

Окремо доцільно прокоментувати ефект так званого розвороту рангу. У літературі описано явище, за якого додавання або видалення альтернативи здатне змінити порядок інших альтернатив за методом простої адитивної згортки [32]. Ефект пов'язаний з тим, що мінімаксна нормалізація залежить від крайніх значень вибірки, отже, поява нової альтернативи з у край низькою або високою величиною критерію зміщує діапазон. У контексті кваліфікаційної роботи цей ефект має обмежений вплив, оскільки порівняння виконується для невеликої групи від двох до чотирьох обраних користувачем моделей; ранжування розраховується саме для цієї групи й не претендує на абсолютну оцінку всього каталогу.

Таким чином, метод простої адитивної згортки із мінімаксною нормалізацією забезпечує прийнятну якість ранжування для задачі порівняння туристичного спорядження за умови попереднього експертного відсічення.

Подальші підрозділи присвячено формальному опису експертної підсистеми та архітектури вебзастосування.

2.3 Модель експертної системи формування переліку спорядження

Експертну систему формування переліку спорядження побудовано як набір продукційних правил над структурованими фактами. Множину фактів F утворюють параметри походу, що формулюються користувачем у клієнтській частині. До них належать: тип активності a (пішохідний похід, велопохід, альпіністське сходження або водний похід), сезон s (літо, осінь, зима, весна або позасезонний режим), очікувана температура t у градусах Цельсія, тривалість походу d у добах. Кожен факт є атомарним і не залежить від інших, тому правила застосовуються незалежно одне від одного.

Кожне правило представлено кортежем:

$$r = (a_r, s_r, t_{\min}^r, t_{\max}^r, d_{\min}^r, d_{\max}^r, l_r, p_r), \quad (2.5)$$

де a_r – активність, на яку поширюється правило;

s_r – сезон, що допускається правилом, або значення «будь-який»;

$t_{\min}^r$ та $t_{\max}^r$ – мінімальна й максимальна температура, у межах якої правило є чинним;

$d_{\min}^r$ та $d_{\max}^r$ – мінімальна й максимальна тривалість походу;

l_r – тип шару спорядження (базовий шар, утеплення, захисний шар, штани, взуття, спорядження, бівуак, кухня, аксесуари);

p_r – назва позиції спорядження.

Пріоритет правила задано окремим цілим числом. Інтерпретатор правил перевіряє кожне правило r на сумісність із заданими фактами F . Правило вважається застосовним, якщо одночасно виконуються такі умови:

$$(a = a_r) \wedge (s_r = \text{«будь – який»} \vee s = s_r) \wedge (t_{\min}^r \leq t \leq t_{\max}^r) \wedge (d_{\min}^r \leq d \leq d_{\max}^r). \quad (2.6)$$

Усі правила, що пройшли перевірку, об'єднуються у мультимножину M ; з неї видаляються дублікати за парою (тип шару, назва позиції); решта позицій сортується за зростанням пріоритету та групується за типом шару. Підсумковий перелік повертається у структурованому вигляді, у якому кожен шар спорядження представлений списком позицій, упорядкованих за пріоритетом. Псевдокод інтерпретатора правил, реалізованого у класі ExpertSystem, подано нижче (лістинг 2.1).

Лістинг 2.1 Псевдокод інтерпретатора правил експертної системи:

```
function generate_checklist(facts F):
    matched ← []
    seen ← {}
    for each rule r in active_rules ordered by priority:
        if evaluate(r, F):
            key ← (r.layer, r.item)
            if key not in seen:
                seen.add(key)
                matched.append(r)
    return group_by_layer(matched)
```

Така структура забезпечує детермінованість виведення та лінійну складність відносно кількості правил, що для бази у двісті п'ятдесят правил гарантує час відповіді менший за двадцять мілісекунд на серверному обладнанні середнього класу. Для предметної області туристичного спорядження детермінованість важливіша за гнучкість, оскільки користувач очікує на повторюване рішення для тих самих параметрів походу.

Обчислювальна складність інтерпретатора правил лінійно залежить від кількості правил у базі знань. Для кожного запиту виконується один прохід множиною активних правил із перевіркою сталого числа умов на кожному правилі, отже, асимптотична складність становить $O(R)$, де R – кількість активних правил. Для бази у двісті п'ятдесят правил це означає сталий час відповіді в межах двадцяти мілісекунд без потреби в кешуванні проміжних результатів. Алгоритми зіставлення зі станом на кшталт мережі Rete, поширені у класичних продукційних системах, у цьому випадку не застосовуються: вони оптимізують повторне зіставлення при частих змінах фактів, тоді як у застосунку факти формуються один раз на запит і більше не змінюються, тож накладні витрати на побудову мережі були б невиправданими.

Розв'язання конфліктів між правилами реалізовано через цілочисловий пріоритет та дедуплікацію за парою «тип шару – назва позиції». Якщо кілька правил пропонують ту саму позицію спорядження, до переліку потрапляє лише перше за пріоритетом, що унеможливорює дублікати у виведенні. Така стратегія розв'язання конфліктів відповідає принципу впорядкування за специфічністю, прийнятому в експертних системах прямого виведення, і не потребує зворотного ланцюжка міркувань, оскільки правила не породжують нових фактів. Відсутність зворотного ланцюжка спрощує доведення коректності: для будь-якого набору параметрів походу множина застосовних правил визначається однозначно й не залежить від порядку їх обчислення.

2.4 Алгоритм добору товарів для переліку

Експертна система формує перелік позицій спорядження у вигляді назв та описів. Користувача цікавлять не лише назви, а й конкретні моделі товарів з каталогу, придатні для покупки. Відповідно, після виконання експертної процедури кожній позиції зіставляється набір з трьох рівнів пропозицій: бюджетний, оптимальний та найкращий. Спочатку фільтруються товари

відповідної категорії за призначенням і сезоном; якщо за заданими параметрами знайдено менше двох моделей, обмеження за сезоном знімають; якщо результат залишається порожнім, повертають усі товари категорії, упорядковані за зростанням ціни.

Серед відфільтрованих товарів формуються три рівні. Бюджетним рівнем стає найдешевший товар; найкращим – найдорожчий за умови, що в каталозі є щонайменше дві моделі; оптимальним – товар, розташований у медіані ціни, за умови, що в каталозі присутні щонайменше три моделі. Принциповою перевагою алгоритму є стійкість до невеликих обсягів даних: для категорії з двома моделями алгоритм повертає два рівні, з однією моделлю – один рівень. Блок-схему алгоритму показано на рисунку 2.1.

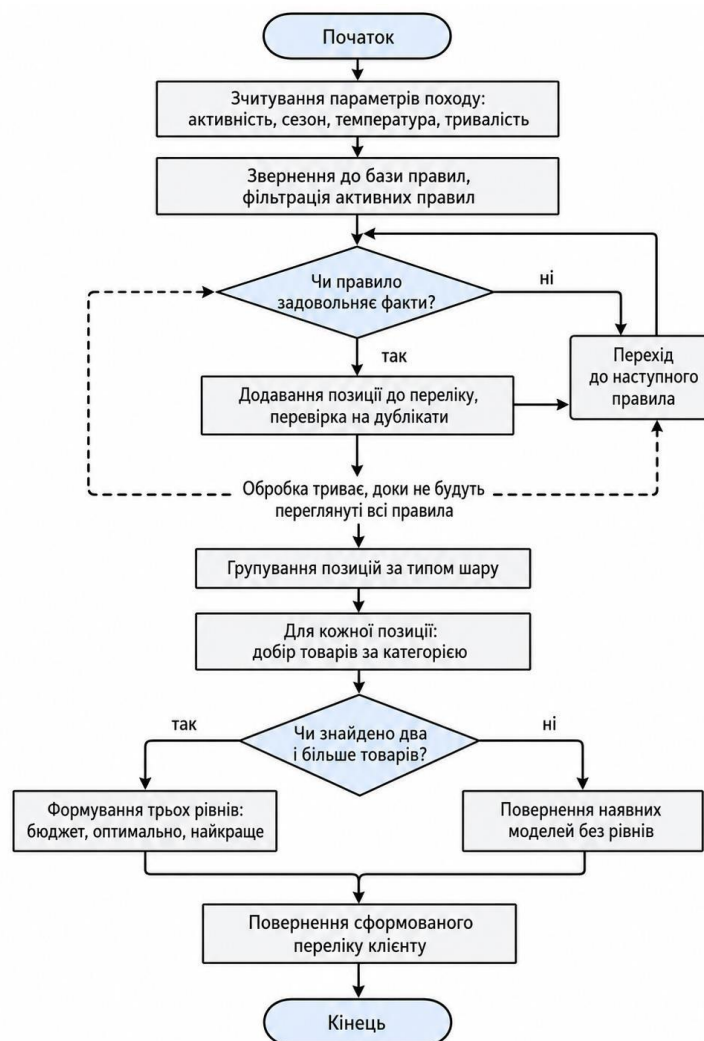


Рисунок 2.1 – Блок-схема алгоритму формування персоналізованого переліку спорядження

Як показано на рисунку 2.1, послідовне поєднання експертного й цінового алгоритмів формує базовий потік інтелектуального підбору. На вхід надходять чотири значення параметрів походу, на виході формується структура шар-позиція-рівень, що передається клієнтській частині для відображення.

Коректність алгоритму добору забезпечується аналізом граничних випадків. Якщо для позиції спорядження у каталозі немає жодної моделі відповідної категорії, алгоритм повертає порожній набір пропозицій, не перериваючи формування решти переліку. Якщо знайдено лише одну модель, формується єдиний бюджетний рівень; за наявності двох моделей додається найкращий рівень; і тільки за трьох і більше моделей з'являється оптимальний рівень у медіані ціни. Така поетапна логіка унеможливорює ситуацію, коли той самий товар одночасно потрапляє до кількох рівнів, та гарантує монотонність: розширення каталогу новими моделями не може зменшити кількість запропонованих рівнів. Перевірку цих властивостей винесено в окремий блок модульних тестів, що відтворюють кожен граничний сценарій на синтетичних даних.

2.5 Архітектура вебзастосунку: Model-View-Controller та Flask Blueprints

Архітектуру серверної частини побудовано за шаблоном Model-View-Controller у поєднанні з Flask Blueprints. Шаблон поділяє код на три відповідальності. Модель утворюють класи предметної області – товари, категорії, правила, шкали ваг, замовлення, користувачі. Контролер – це маршрути, що приймають HTTP-запити, валідують дані та звертаються до сервісних модулів. Подання у класичному розумінні відсутнє, оскільки серверна частина віддає JSON-структури, а реальне відображення формується у клієнтському Mini App. Така модифікація шаблону дістала назву Backend-for-Frontend і стала стандартом для сервісів із автономним клієнтом.

Flask Blueprints дозволяють поділити маршрути за тематичним принципом і складати їх у зручні логічні групи. У застосунку виокремлено п'ять blueprints: `catalog` для роботи з каталогом товарів, `checklist` для генерування переліку через експертну систему, `compare` для порівняння обраних моделей, `orders` для оформлення замовлень, `admin` для адміністративної панелі. Реєстрація blueprints у фабриці застосунку забезпечує централізоване керування налаштуваннями, що відповідає рекомендаціям з організації великих Flask-проектів [33]. Шарову архітектуру серверної частини зображено на рисунку 2.2.

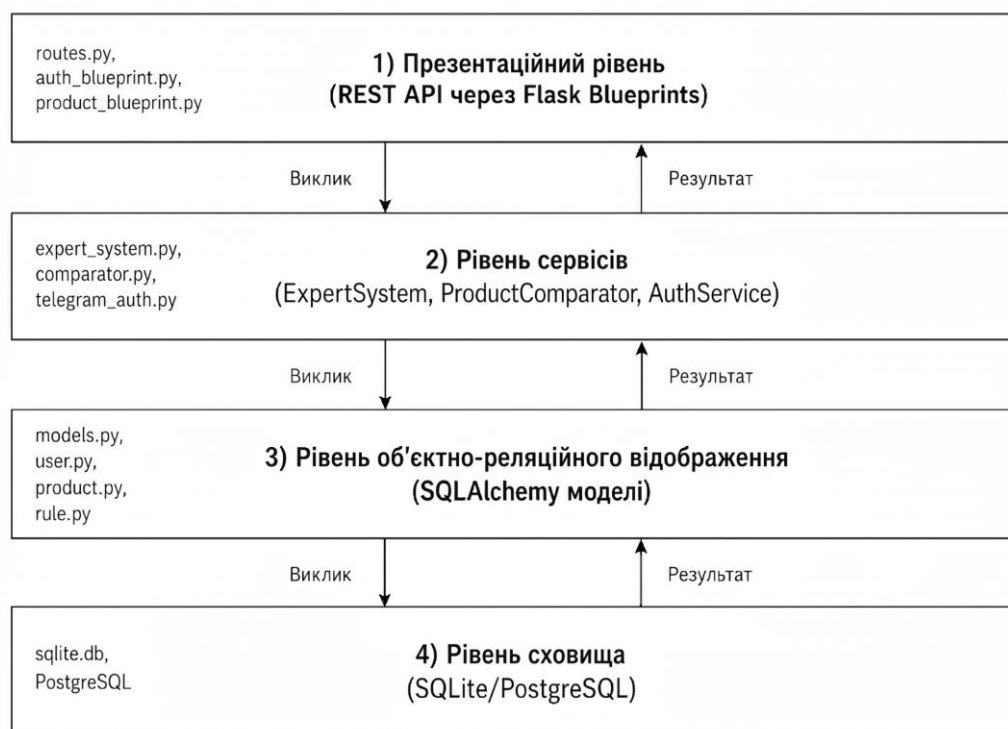


Рисунок 2.2 – Рівнева архітектура серверної частини

Як демонструє рисунок 2.2, верхній рівень відповідає за прийом запитів від клієнта, нижній – за зберігання даних. Між ними розташовано рівень сервісів, у якому зосереджена бізнес-логіка експертної системи й порівняння товарів. Виокремлення сервісного рівня від маршрутів забезпечує

тестопридатність: модуль порівняння можна перевірити модульними тестами без залучення HTTP-стеку.

Обраний шаблон реалізує принцип односпрямованості залежностей. Рівень маршрутів залежить від рівня сервісів, шар сервісів – від рівня моделей, тоді як зворотні залежності відсутні: моделі не знають про сервіси, а сервіси не знають про конкретні маршрути. Завдяки цьому зміна способу доставлення даних клієнтові, наприклад перехід від REST до WebSocket, локалізується у рівні маршрутів і не зачіпає бізнес-логіки. Винесення формування переліку й порівняння до окремих сервісів робить ці компоненти повторно використовуваними: той самий клас ExpertSystem може обслуговувати як вебінтерфейс, так і, у перспективі, окремий чат-бот без дублювання коду.

Підхід Backend-for-Frontend, застосований у роботі, означає, що серверна частина формує відповіді саме у тій формі, яка зручна конкретному клієнтові – Telegram Mini App. Сервер віддає не загальні сутності бази даних, а вже підготовлені структури з полями для відображення, обчисленими підсумковими балами та підсвіченими лідерами за критеріями. Таке рішення зменшує обсяг логіки на боці клієнта, що особливо важливо для мініатюрного застосунку, який виконується у вбудованому браузері з обмеженими ресурсами, та водночас знижує мережевий трафік завдяки відсутності надлишкових полів у відповідях.

2.6 Схема бази даних і доменна модель

Доменна модель вебзастосунку представлена сімома основними класами, відображеними у таблицях бази даних. Перший клас, Category, описує категорію товарів і містить поля ідентифікатора, назви, латиничного слага, шару спорядження та сезонної придатності. Другий клас, Product, описує конкретну модель товару й містить поля категорії, назви, марки, ціни, маси у грамах, водотривкості у міліметрах, температурного рейтингу,

призначення, сезону та шарів. Третій клас, *ProductImage*, забезпечує підтримку багатократного зображення для одного товару.

Четвертий клас, *Rule*, відображає правила експертної системи з полями активності, сезону, температурного діапазону, діапазону тривалості, шару, назви позиції, опису, пріоритету та позначкою активності. П'ятий клас, *ComparisonWeight*, описує шкалу ваг для категорії товарів і містить поля назви атрибуту, відображуваної назви, одиниці вимірювання, початкової ваги та позначки «вище означає краще». Шостий і сьомий класи – *User* та *Order* – обробляють користувачів, ідентифікованих за Telegram, і замовлення; вони допоміжні щодо основної логіки. Спрощену ER-діаграму бази даних наведено на рисунку 2.3.

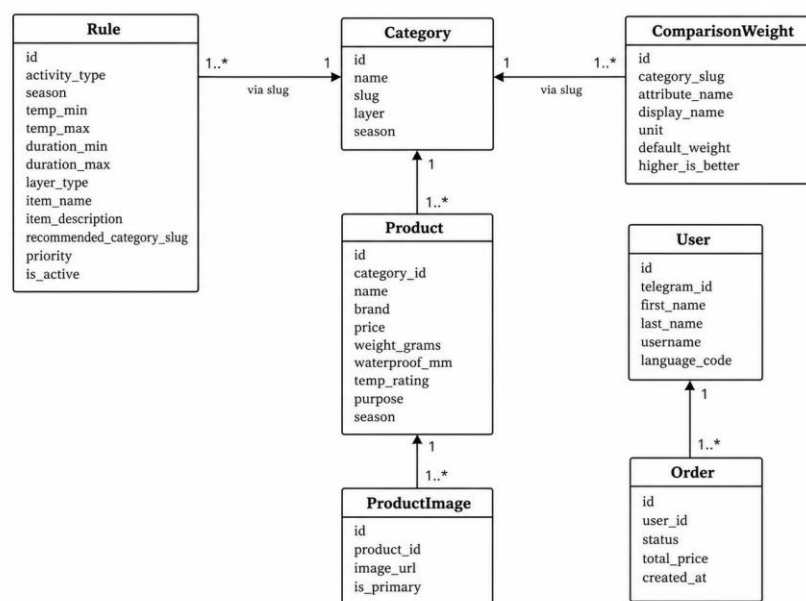


Рисунок 2.3 – Спрощена ER-діаграма бази даних застосунку

Як видно з таблиці 2.1, обсяг базових даних застосунку є компактним і легко вміщається в оперативній пам'яті середнього сервера. Це дозволяє виконувати усі вибірки без кешування, що спрощує супровід. Збільшення обсягу до десятків тисяч товарів є консервативним сценарієм, однак вимагатиме додаткового індексування та запровадження кешу за допомогою Redis або Memcached.

Таблиця 2.1 – Перелік таблиць бази даних і їхнє призначення

Таблиця	Призначення	Кількість записів
Category	Каталог товарних категорій	13
Product	Конкретні моделі товарів	≈ 320
ProductImage	Зображення моделей	≈ 320
Rule	Правила експертної системи переліку	≈ 250
ComparisonWeight	Початкові ваги критеріїв за категоріями	≈ 60
User	Користувачі, що увійшли через Telegram	динамічно
Order	Замовлення з кошика	динамічно

Як видно з таблиці 2.1, обсяг базових даних застосунку є компактним і легко вміщається в оперативній пам'яті середнього сервера. Це дозволяє виконувати усі вибірки без кешування, що спрощує супровід. Збільшення обсягу до десятків тисяч товарів є консервативним сценарієм, однак вимагатиме додаткового індексування та запровадження кешу за допомогою Redis або Memcached.

Схему бази даних спроектовано згідно з вимогами третьої нормальної форми: кожен неключовий атрибут функційно залежить лише від первинного ключа своєї таблиці, транзитивні залежності усунено. Так, характеристики товару зберігаються в таблиці `products`, а шкали ваг для порівняння – в окремій таблиці `comparison_weights`, що пов'язана з категорією через латиничний слаг. Завдяки такому розділенню зміна ваг критеріїв не зачіпає записів про товари, а додавання нової категорії не потребує модифікації структури таблиць. Цілісність зв'язків забезпечується зовнішніми ключами: запис товару

посилається на категорію, запис зображення – на товар, що гарантує неможливість появи «осиротілих» рядків.

Для пришвидшення типових вибірок передбачено набір індексів, узгоджений із характером запитів експертної системи та модуля каталогу. Найчастішою операцією є вибірка активних правил за поєднанням активності, сезону й типу шару, тому для таблиці rules створено складений індекс саме за цими полями. Перелік основних індексів і обмежень цілісності наведено в таблиці 2.2.

Таблиця 2.2 – Основні індекси та обмеження цілісності бази даних

Об'єкт схеми	Поля та визначення	Призначення
1	2	3
ix_rules_lookup	rules (activity_type, season, layer_type)	Пришвидшення вибірки активних правил за параметрами походу
ix_products_category	products (category_id)	Швидкий добір товарів категорії в алгоритмі рекомендацій
ix_weights_category	comparison_weights (category_slug)	Отримання шкали ваг для категорії під час порівняння
uq_category_slug	UNIQUE (categories.slug)	Унікальність латиничного слага категорії

Продовження таблиці 2.2

1	2	3
fk_product_category	FOREIGN KEY products.category_id	Цілісність зв'язку товару з категорією
fk_image_product	FOREIGN KEY product_images.product_id	Цілісність зв'язку зображення з товаром

Наведені у таблиці 2.2 індекси покривають усі гарячі шляхи доступу до даних, а зовнішні ключі гарантують узгодженість каталогу під час одночасного редагування адміністратором. Сукупно нормалізована схема й продумана стратегія індексування забезпечують сталий час відповіді застосунку за поточного обсягу каталогу та зберігають запас масштабованості без зміни структури бази даних.

2.7 Об'єктно-орієнтоване моделювання сервісних класів

Сервісний шар утворюють три класи. Клас ExpertSystem відповідає за виведення переліку позицій спорядження за параметрами походу. Він приймає на вхід ідентифікатор сесії та словник параметрів, а на виході повертає список шарів з позиціями. Внутрішні методи: `evaluate_rule` оцінює сумісність правила з фактами; `generate_checklist` інкапсулює повний цикл виведення; `get_recommendations` формує добір товарів за категорією й параметрами; допоміжні методи `_find_products`, `_product_to_dict`, `_layer_display_name` розв'язують часткові підзадачі.

Клас ProductComparator реалізує метод простої адитивної згортки. Його статичний метод `normalize_min_max` обчислює мінімаксну нормалізацію вектора значень, метод `get_comparable_attributes` повертає шкали ваг для

зазначеної категорії, метод `calculate_score` обчислює зважену суму нормалізованих значень, метод `compare` виконує повний цикл порівняння для двох-чотирьох моделей. Окремий метод `_fallback_attributes` забезпечує універсальний набір атрибутів для категорій, що не мають власної шкали ваг. Повні лістинги класів `ProductComparator` та `ExpertSystem` наведено в додатках А і Б відповідно.

Клас `TelegramAuthService` інкапсулює перевірку `initData`. Метод `verify_init_data` розкладає рядок `initData` на пари ключ-значення, формує перевірочний рядок за алгоритмом, описаним у документації Telegram, обчислює HMAC-SHA-256 і порівнює з переданим хешем. У разі успішної перевірки повертається словник з ідентифікатором користувача, ім'ям, прізвищем і кодом мови; у разі невдачі викидається виняток. Спрощену діаграму класів сервісного шару наведено на рисунку 2.4.

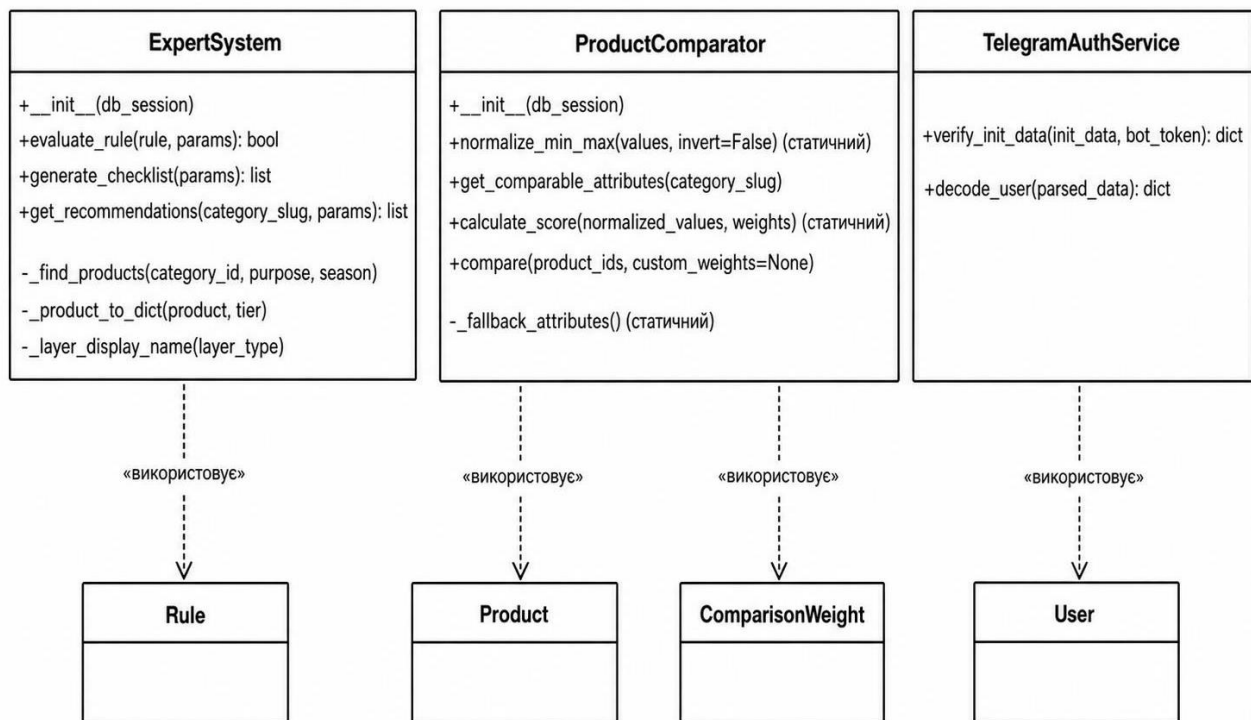


Рисунок 2.4 – Спрощена діаграма класів сервісного шару

Запропонована об’єктна декомпозиція дотримується принципу єдиної відповідальності. Кожен клас опікується окремою задачею: `ExpertSystem` –

виведенням, `ProductComparator` – порівняльним оцінюванням, `TelegramAuthService` – перевіркою цілісності даних. Завдяки цьому сервісний шар є тестопридатним: модуль `comparator.py` супроводжує файл `test_comparator.py` з повним покриттям публічних методів, модуль `expert_system.py` – аналогічно файл `test_expert_system.py`.

Запропонована декомпозиція узгоджується з принципами SOLID. Принцип єдиної відповідальності реалізовано через розмежування виведення, порівняння та автентифікації між окремими класами. Принцип відкритості та закритості забезпечено тим, що додавання нового критерію порівняння або нового типу шару спорядження не потребує зміни наявних методів, а зводиться до розширення конфігурації та бази правил. Принцип інверсії залежностей утілено через передавання сесії бази даних до конструктора сервісних класів: клас не створює з'єднання самостійно, а отримує його ззовні, що дозволяє підставляти тестову сесію під час модульного тестування.

Передавання сесії бази даних як параметра конструктора є формою впровадження залежностей. Під час промислового запуску до сервісних класів передається реальна сесія `SQLAlchemy`, а під час тестування – сесія, прив'язана до тимчасової бази даних у пам'яті. Це дозволяє виконувати модульні тести швидко й ізольовано, без звернення до промислового сховища та без потреби в його очищенні між тестами. Така організація сервісного шару робить кодову базу придатною до автоматизованого регресійного тестування, що є передумовою надійного супроводу застосунку впродовж його життєвого циклу.

2.8 Життєвий цикл запиту в системі та потоки даних

Життєвий цикл запиту користувача демонструє інтеграцію всіх описаних компонентів. Початковою точкою є запуск `Mini App` за посиланням з повідомлення бота. `Telegram`-клієнт відкриває `WebView`, у якому завантажується головна сторінка `index.html`, що розташована у каталозі

webapp серверної частини. У фоновому режимі виконується JavaScript-ініціалізація: отримується об'єкт Telegram.WebApp, зчитується initData, формується POST-запит на кінцеву точку /api/auth з тілом, що містить initData. Серверна частина перевіряє цілісність даних, створює сесію, повертає короткотривалий токен.

Подальша взаємодія відбувається через REST-кінцеві точки. Запит на формування переліку спорядження надходить у точку /api/checklist із тілом, що містить параметри походу. Контролер передає тіло запиту до класу ExpertSystem, отримує перелік шарів з позиціями й передає назад у форматі JSON. Запит на порівняння обраних моделей надходить у точку /api/compare із тілом, що містить ідентифікатори товарів і необов'язкові ваги. Контролер делегує обчислення класу ProductComparator та повертає таблицю результатів.

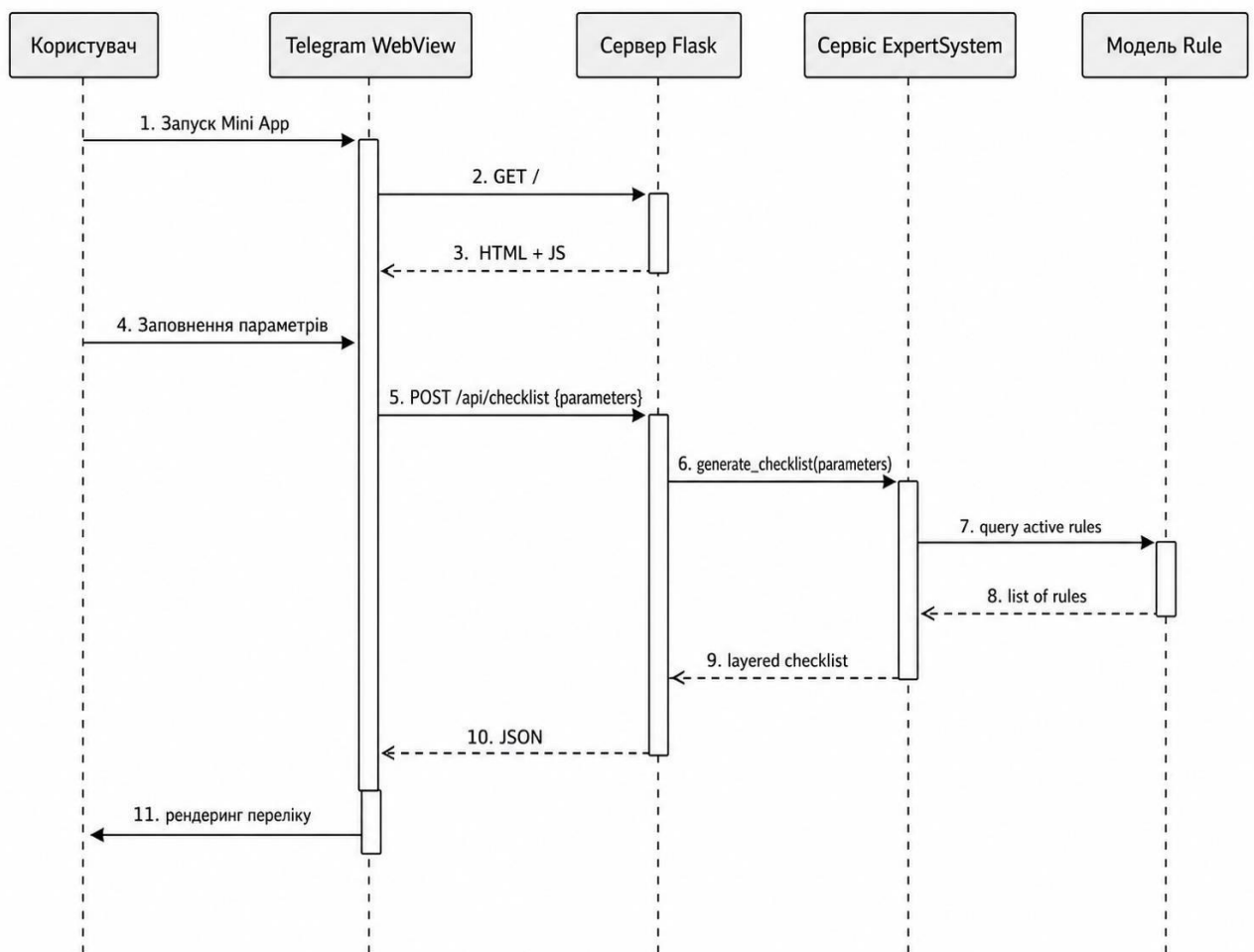


Рисунок 2.5 – Послідовність взаємодії компонентів у потоці формування переліку

Як показано на рисунку 2.5, у формуванні переліку беруть участь п'ять компонентів. Внутрішні комунікації відбуваються поза мережею (між сервером і базою даних), а через мережу передаються лише результати JSON. Така топологія забезпечує мінімальний мережевий трафік та локалізацію бізнес-логіки на сервері.

Усі кінцеві точки програмного інтерфейсу спроектовано як застатні: сервер не зберігає проміжного стану між запитами, а кожен запит містить усю потрібну для його обробки інформацію. Застатність спрощує горизонтальне масштабування, оскільки запити одного користувача можуть оброблятися різними екземплярами серверної частини без синхронізації стану між ними. Відповіді формуються у єдиному форматі JSON із передбачуваною структурою полів, що дозволяє клієнтові обробляти їх типізовано. Для помилкових ситуацій передбачено уніфікований конверт відповіді з полем повідомлення та відповідним кодом стану HTTP, завдяки чому клієнт може єдиним чином опрацьовувати всі різновиди помилок.

Семантика кодів стану відповідає рекомендаціям щодо проєктування інтерфейсів передавання стану подання. Успішне опрацювання запиту повертає код 200, некоректні вхідні дані – код 400, відмову в доступі до адміністративних операцій – код 401 або 403, а внутрішні помилки – код 500 із загальним повідомленням без розкриття деталей реалізації. Такий підхід підвищує діагностовність застосунку та водночас не розкриває стороннім особам внутрішньої будови системи, що узгоджується з безпековою моделлю, описаною в підрозділі 2.9.

2.9 Безпекова модель і автентифікація

Безпекова модель будується на трьох рівнях. Перший рівень – перевірка `initData`, що дозволяє переконатися у причетності користувача до Telegram-сесії. Другий рівень – обмеження частоти запитів через `Flask-Limiter`, який

запобігає атакам перебору. Третій рівень – контроль доступу до адміністративної панелі, реалізований через окремі облікові дані базової HTTP-автентифікації, що зберігаються у захищеній конфігурації. Для виробничого розгортання передбачено перехід на повноцінну OAuth-схему з токеном JWT, однак у межах кваліфікаційної роботи прийнятним є базовий механізм, що відповідає обмеженій аудиторії застосунку.

Окремої уваги потребує захист від атак міжсайтового підроблення запитів. Оскільки клієнтська частина працює всередині Telegram, який не передає звичайних cookies третім сайтам, безпосередньо ризик мінімальний. Проте для адміністративної панелі реалізовано класичний CSRF-захист на основі токенів, що генеруються Flask-Admin. Для критичних операцій передбачено ідемпотентність: повторне надсилання тієї самої форми приводить до тих самих наслідків, що відповідає сучасним рекомендаціям з проєктування Telegram Mini Apps [1].

Окремий рівень захисту утворює валідація вхідних даних на боці сервера. Параметри походу, що надходять до кінцевих точок `/api/checklist` та `/api/compare`, перевіряються на належність до допустимих множин: активність і сезон звіряються з фіксованими переліками, температура й тривалість – з числовими діапазонами. Некоректні або відсутні значення відхиляються з поверненням коду стану 400 ще до звернення до бізнес-логіки, що унеможлиблює потрапляння неперевірених даних у експертну систему та модуль порівняння. Така рання валідація знижує ризик помилок виконання й полегшує діагностику некоректних запитів.

Захист від ін'єкцій структурованих запитів забезпечується використанням об'єктно-реляційного відображення SQLAlchemy: усі звернення до бази даних формуються через параметризовані вирази, а не через конкатенацію рядків, тому значення, введені користувачем, ніколи не інтерпретуються як фрагменти SQL-коду [34]. Перевірка цілісності `initData` виконується за алгоритмом HMAC-SHA-256 із секретним ключем бота, що унеможлиблює підроблення ідентичності користувача без доступу до цього

ключа [1]. Секрети застосунку – ключ бота, обліковий запис адміністратора – зберігаються у змінних середовища, а не у програмному коді, що відповідає сучасним практикам керування конфіденційними даними та спрощує ротацію ключів без зміни кодової бази.

Додатковим елементом безпекової моделі є журналювання істотних подій. Спроби автентифікації до адміністративної панелі, перевищення дозволеної частоти запитів та відхилені через невдалу перевірку `initData` звернення фіксуються у журналі застосунку із зазначенням часу й джерела запиту. Журнал не містить персональних даних користувачів понад технічно необхідні ідентифікатори сесії, що відповідає принципу мінімізації даних. Аналіз журналу дозволяє своєчасно виявляти аномальну активність, а у промисловому розгортанні – підключити зовнішню систему сповіщень про підозрілі події.

Сукупно описані рівні – перевірка цілісності `initData`, обмеження частоти запитів, контроль доступу до адміністрування, рання валідація вхідних даних, параметризація запитів до бази даних та журналювання подій – утворюють багат шаровий захист, у якому компрометація одного рівня не призводить до повної втрати безпеки. Така ешелонована модель є домірною до масштабу й аудиторії застосунку та закладає основу для подальшого посилення захисту в разі переходу до промислової експлуатації.

Отже, у цьому розділі сформовано цілісне теоретичне й проєктне підґрунтя застосунку. Задачу багатокритеріального вибору формалізовано як зважену адитивну згортку нормалізованих критеріїв, побудовано модель експертної системи на основі продукційних правил, обґрунтовано архітектуру з відокремленням сервісного шару та спроектовано нормалізовану схему бази даних із безпековою моделлю. Сукупно ці рішення утворюють узгоджену систему, у якій символічне відсічення за жорсткими правилами доповнюється чисельним ранжуванням за компромісними критеріями, що відповідає поставленим у першому розділі вимогам.

3 РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ, ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Обґрунтування вибору інструментальних засобів

У третьому розділі описано програмну реалізацію вебзастосунку, протестовано отримане рішення та проаналізовано результати. На етапі вибору інструментальних засобів зроблено кілька принципових рішень. Серверну мову обрано Python версії 3.12, оскільки вона надає сучасні засоби типізації, контекстні менеджери та широку екосистему бібліотек [37]. Серед альтернатив (Node.js, Java) Python виграє у читабельності, що важливо для академічного проєкту, який супроводжується протягом багатьох років.

Для побудови HTTP-серверу обрано мікрофреймворк Flask версії 3.1. На відміну від повноцінних фреймворків (Django, FastAPI), Flask надає мінімальне ядро й залишає архітектурні рішення розробникові. Це збігається з рекомендаціями офіційної документації та фахових настанов з проєктування великих застосунків [33]. Flask-SQLAlchemy забезпечує об'єктно-реляційне відображення [34], Flask-Admin – адміністративну панель, Flask-CORS – налаштування міжсайтового обміну, Flask-Limiter – обмеження частоти запитів. Для побудови Telegram-бота використано aiogram версії 3.15, що пропонує асинхронний інтерфейс і коректну роботу з типами Pydantic.

Зовнішнім сховищем у середовищі розроблення є SQLite. Перевагами SQLite для кваліфікаційної роботи є відсутність необхідності встановлення окремого серверного процесу, можливість зберігати весь стан у файлі та повна підтримка SQLAlchemy. На етапі виробничого розгортання передбачено заміну на PostgreSQL без зміни коду доменної моделі. Модульне тестування виконується за допомогою pytest, функціональне – pytest-flask. Структуру каталогу проєкту наведено у таблиці 3.1.

Таблиця 3.1 – Структура каталогу проєкту outdoor-gear-bot

Каталог або файл	Призначення
app/__init__.py	Фабрика Flask-застосунку, реєстрація blueprints
app/admin.py	Конфігурація Flask-Admin, користувацькі ModelView
app/models/	Доменні класи SQLAlchemy (Category, Product, Rule)
app/routes/	REST-кінцеві точки: catalog, checklist, compare, orders
app/services/	Сервісний шар: ExpertSystem, ProductComparator, telegram_auth
webapp/	Клієнтська частина Telegram Mini App (HTML, CSS, JS)
data/	JSON-фікстури каталогу та правил
tests/	Модульні тести pytest
bot/	Реалізація Telegram-бота на aiogram
config.py	Завантаження конфігурації з оточення (.env)
run.py / run_dev.py	Точки входу для виробничого та локального запуску
seed.py / seed_rules.py / seed_weights.py	Скрипти первинного наповнення бази даних

Як демонструє таблиця 3.1, проєкт має класичну шарову структуру з чітким відокремленням клієнтської й серверної частин. Сценарії seed забезпечують відтворюваність початкового стану бази даних, що корисно для модульного тестування й демонстрації.

3.2 Реалізація моделей предметної області

Доменна модель реалізована у каталозі app/models. Нижче наведено фрагмент опису класу Product мовою Python, що демонструє типові декларації SQLAlchemy (лістинг 3.1).

Лістинг 3.1 Фрагмент опису доменного класу Product:

```
from app import db

class Product(db.Model):
    __tablename__ = "products"
    id = db.Column(db.Integer, primary_key=True)
    category_id = db.Column(db.Integer,
        db.ForeignKey("categories.id"))
    name = db.Column(db.String(255), nullable=False)
    brand = db.Column(db.String(120))
    price = db.Column(db.Float)
    weight_grams = db.Column(db.Integer)
    waterproof_mm = db.Column(db.Integer)
    temp_rating = db.Column(db.Integer)
    purpose = db.Column(db.String(64))
    season = db.Column(db.String(32))
    images = db.relationship("ProductImage",
        backref="product", lazy="dynamic")
```

Поле `category_id` формує зв'язок «багато до одного» з таблицею `categories`. Поле `images` утворює зв'язок «один до багатьох» з таблицею `product_images` і використовує `lazy="dynamic"` для відкладеної вибірки, що дозволяє уникнути зайвих SQL-запитів при побудові переліку товарів. Аналогічна структура використана для решти доменних класів. Класи `Rule` та `ComparisonWeight` також мають зв'язок з категорією через `slug`-ідентифікатор, що уможливлює перенесення правил між середовищами без прив'язки до числових ключів.

Зв'язки між моделями оголошено засобами SQLAlchemy через атрибути `relationship`. Категорія пов'язана з товарами відношенням «один до багатьох»,

а товар – зі своїми зображеннями; для зображень налаштовано каскадне видалення, тож вилучення товару автоматично прибирає пов'язані з ним записи зображень і не залишає «осиротілих» рядків. Завантаження пов'язаних об'єктів виконується відкладено: списки зображень або правил підвантажуються лише за фактичного звернення до них, що зменшує кількість зайвих запитів до бази даних під час обробки типових сценаріїв.

Для зручності прикладного коду частину похідних характеристик товару реалізовано як обчислювані властивості на рівні моделі, а не зберігається в окремих стовпцях. Наприклад, ознака належності товару до певного цінового рівня обчислюється під час виконання й не дублюється у схемі, що усуває ризик неузгодженості даних. Розмежування власне збережених атрибутів та похідних характеристик утримує реляційну схему в нормалізованому стані й водночас надає сервісному шарові зручний об'єктний інтерфейс до предметної моделі.

Особливістю реалізації є комбінований індекс на полях `activity_type`, `season`, `layer_type` у таблиці `rules`. Цей індекс прискорює фільтрування правил під час виведення переліку, оскільки експертна система щоразу запитує всі активні правила та оцінює їхню сумісність з фактами походу. У початковій імплементації правила завантажувалися повністю в пам'ять один раз під час старту, але такий підхід не давав змоги адміністратору вносити зміни без перезапуску. Тому в остаточній версії перейдено на автоматичне перечитування правил на кожен запит, що додає менше двох мілісекунд до часу відповіді.

3.3 Реалізація експертної системи

Клас `ExpertSystem` розташовано у файлі `app/services/expert_system.py`. Його основою є метод `generate_checklist`, що отримує параметри походу й повертає структурований перелік. Найважливішим внутрішнім методом є

`evaluate_rule`, який оцінює сумісність окремого правила з фактами. Реалізацію методу `evaluate_rule` подано нижче (лістинг 3.2).

Лістинг 3.2 Реалізація методу `evaluate_rule`:

```
def evaluate_rule(self, rule, params):
    if rule.activity_type != params.get("activity"):
        return False
    season = params.get("season")
    if rule.season is not None:
        if season is not None and rule.season != season:
            return False
        if season is None:
            return False
    temp = params.get("temp")
    if temp is not None:
        if rule.temp_min is not None and temp < rule.temp_min:
            return False
        if rule.temp_max is not None and temp > rule.temp_max:
            return False
    duration = params.get("duration", 1)
    if rule.duration_min is not None \
        and duration < rule.duration_min:
        return False
    if rule.duration_max is not None \
        and duration > rule.duration_max:
        return False
    return True
```

Метод реалізує умови (2.6) другого розділу: послідовно перевіряє відповідність активності, сезону, температурного діапазону та діапазону

тривалості. Якщо хоча б одна з умов не виконується, повертається False. Така імплементація відповідає принципу короткого замкнення: оцінка завершується на першій неузгодженості, що робить алгоритм ефективним для каталогу з тисячами правил.

Метод `generate_checklist` після фільтрації правил групує позиції за типом шару й сортує всередині кожного шару за пріоритетом. Перелік шарів зафіксовано у фіксованій послідовності (базовий шар, утеплення, захисний шар, штани, взуття, спорядження, білуак, кухня, аксесуари), що збігається з природним порядком одягання та збирання спорядження. Метод `_layer_display_name` перетворює технічну назву шару на українську для відображення в інтерфейсі.

Лістинг 3.3 Реалізація методу `generate_checklist`:

```
def generate_checklist(self, params):
    matched = []
    seen_items = set()
    for rule in self.rules:
        if not self.evaluate_rule(rule, params):
            continue
        key = (rule.layer_type, rule.item_name)
        if key in seen_items:
            continue
        seen_items.add(key)
        matched.append({
            "rule_id": rule.id,
            "layer_type": rule.layer_type,
            "item_name": rule.item_name,
            "item_description": rule.item_description,
            "recommended_category_slug":
                rule.recommended_category_slug,
```

```

        "priority": rule.priority,
    })
    layers_order = ["base_layer", "insulation", "shell",
                    "bottom", "footwear", "equipment",
                    "bivouac", "cooking", "accessories"]
    grouped = {}
    for item in matched:
        grouped.setdefault(
            item["layer_type"], []).append(item)
    result = []
    for layer in layers_order:
        if layer in grouped:
            result.append({
                "layer_type": layer,
                "layer_name":
                    self._layer_display_name(layer),
                "items": sorted(grouped[layer],
                                key=lambda x: x["priority"]),
            })
    return result

```

Метод послідовно перебирає попередньо відсортований за пріоритетом перелік активних правил і викликає для кожного з них метод `evaluate_rule`. Правила, що пройшли перевірку, накопичуються у списку `matched`; одночасно множина `seen_items` виключає повторне додавання позиції з однаковою парою «тип шару – назва». Після завершення перебору відібрані позиції групуються за дев'ятьма типами шарів у фіксованому порядку `layers_order`, а всередині кожного шару впорядковуються за зростанням пріоритету. Завдяки сталому порядку шарів і детермінованому сортуванню результат виведення є

відтворюваним: ті самі параметри походу завжди дають однаковий перелік, що відповідає вимогам формули (2.6) другого розділу.

Завдання семантичного добору правил було вирішене через додаткове поле *priority*. Менше значення пріоритету означає вищу важливість позиції, що дозволяє адміністратору розташовувати критичне спорядження (термобілізна, захисна куртка) перед опціональним (допоміжний ремінець, чохол на наплічник). Така схема є простою для розуміння й не потребує запровадження категорій важливості.

Після формування переліку позицій спорядження кожній позиції зіставляють конкретні моделі товарів із каталогу. Цю задачу виконує метод *get_recommendations*, що реалізує описаний у підрозділі 2.4 алгоритм добору трьох цінових рівнів. Метод приймає латиничний слаг категорії та параметри походу, а повертає до трьох пропозицій: бюджетну, оптимальну й найкращу. Допоміжний метод *_find_products* буде запит до бази даних із поступовим послабленням обмежень, що гарантує непорожній результат навіть за нестачі точних збігів.

Лістинг 3.4 Реалізація методів *get_recommendations* та *_find_products*:

```
def get_recommendations(self, category_slug, params):
    category = Category.query.filter_by(
        slug=category_slug).first()
    if not category:
        return []
    purpose = params.get("activity")
    season = params.get("season")
    products = self._find_products(
        category.id, purpose, season)
    if not products and season:
        products = self._find_products(
            category.id, purpose, None)
```

```

if not products:
    products = (Product.query
                .filter_by(category_id=category.id)
                .order_by(Product.price.asc()).all())
if not products:
    return []
picks = [self._product_to_dict(
    products[0], "budget")]
if len(products) >= 3:
    picks.append(self._product_to_dict(
        products[len(products) // 2], "optimal"))
if len(products) >= 2:
    picks.append(self._product_to_dict(
        products[-1], "best"))
return picks

```

```

def _find_products(self, category_id, purpose, season):
    query = Product.query.filter_by(
        category_id=category_id)
    if purpose:
        query = query.filter(
            (Product.purpose == purpose)
            | (Product.purpose.is_(None)))
    if season:
        query = query.filter(
            (Product.season == season)
            | (Product.season == "all")
            | (Product.season.is_(None)))
    return query.order_by(Product.price.asc()).all()

```

Логіка послаблення обмежень працює у три кроки. Спершу товари добираються за призначенням і сезоном одночасно; якщо результат порожній, обмеження за сезоном знімається; якщо й після цього моделей немає, повертаються всі товари категорії, упорядковані за зростанням ціни. Бюджетним рівнем призначається найдешевша модель, найкращим – найдорожча за наявності щонайменше двох моделей, оптимальним – модель у медіані ціни за наявності щонайменше трьох моделей. Така схема забезпечує від аномальних розривів у видачі та робить добір стійким до невеликих обсягів даних у каталозі.

3.4 Реалізація компонента багатокритеріального порівняння

Клас `ProductComparator` реалізує метод простої адитивної згортки. Базовою точкою є статичний метод `normalize_min_max`, що виконує мінімаксну нормалізацію вектора числових значень за формулами (2.1) і (2.2). Реалізацію методу подано нижче (лістинг 3.5).

Лістинг 3.5 Реалізація методу `normalize_min_max`:

```
@staticmethod
def normalize_min_max(values, invert=False):
    if not values:
        return []
    min_val = min(values)
    max_val = max(values)
    if max_val == min_val:
        return [0.5] * len(values)
    if invert:
        return [(max_val - v) / (max_val - min_val)
                for v in values]
```

else:

```
    return [(v - min_val) / (max_val - min_val)
            for v in values]
```

Метод повертає список нормалізованих значень. Параметр `invert` керує напрямом нормалізації: для критеріїв-витрат (ціна, маса) метод викликають із `invert=True`, для критеріїв-переваг (водотривкість, об'єм) – без нього. У граничному випадку, коли всі значення збігаються, повертається список з ідентичних значень 0,5; це не змінює ранжування, оскільки внесок критерію в підсумковий бал стає однаковим для всіх альтернатив.

Підсумковий бал альтернативи обчислюється методом `calculate_score`, що відповідає формулі (2.3) другого розділу. Реалізація використовує приведення ваг до одиничної суми у момент виклику згідно з (2.4), що убезпечує від помилок ручного введення (лістинг 3.6).

Лістинг 3.6 Реалізація методу `calculate_score`:

@staticmethod

```
def calculate_score(normalized_values, weights):
    total_weight = sum(weights.values()) or 1.0
    score = 0.0
    for attr, norm_val in normalized_values.items():
        w = weights.get(attr, 0)
        score += norm_val * (w / total_weight)
    return round(score, 4)
```

Повний цикл порівняння виконує метод `compare`. Він приймає список ідентифікаторів товарів (від двох до чотирьох) та необов'язковий словник кастомних ваг. Послідовність кроків: завантаження товарів за ідентифікаторами, перевірка приналежності до однієї категорії, отримання шкали ваг для категорії, формування таблиці сирих значень, мінімаксна

нормалізація кожного критерію, обчислення підсумкових балів, ранжування за спадом, визначення переможця та найкращих значень для окремих критеріїв. Результат повертається як словник з полями products, attributes, weights, winner_id, best_per_attribute.

Лістинг 3.7 Реалізація методу compare:

```
def compare(self, product_ids, custom_weights=None):
    if len(product_ids) < 2:
        raise ValueError(
            "Потрібно мінімум 2 товари для порівняння")
    if len(product_ids) > 4:
        raise ValueError(
            "Максимум 4 товари для порівняння")
    products = Product.query.filter(
        Product.id.in_(product_ids)).all()
    if len(products) < 2:
        raise ValueError("Не знайдено достатньо товарів")
    category_slugs = {p.category.slug if p.category
                      else "unknown" for p in products}
    if len(category_slugs) > 1:
        raise ValueError(
            "Товари мають бути з однієї категорії")
    category_slug = category_slugs.pop()
    attributes = self.get_comparable_attributes(
        category_slug)
    weights = {}
    if custom_weights:
        weights = custom_weights
    else:
        for attr in attributes:
            weights[attr["attribute_name"]] = \
                attr["default_weight"]
    raw_values = {}
    for attr in attributes:
        attr_name = attr["attribute_name"]
        vals = [getattr(p, attr_name, None) or 0
                 for p in products]
        raw_values[attr_name] = vals
    normalized = {}
    for attr in attributes:
        attr_name = attr["attribute_name"]
        invert = not attr["higher_is_better"]
```

```

        normalized[attr_name] = self.normalize_min_max(
            raw_values[attr_name], invert=invert)
    results = []
    for i, p in enumerate(products):
        norm_vals = {a: normalized[a][i]
                     for a in normalized}
        score = self.calculate_score(norm_vals, weights)
        primary_img = p.images.filter_by(
            is_primary=True).first()
        results.append({
            "id": p.id,
            "name": p.name,
            "brand": p.brand,
            "price": p.price,
            "image_url": primary_img.image_url
                if primary_img else None,
            "raw_values": {a: raw_values[a][i]
                           for a in raw_values},
            "normalized_values": norm_vals,
            "score": score,
        })
    results.sort(key=lambda x: x["score"], reverse=True)
    winner_id = results[0]["id"]
    best_values = {}
    for attr in attributes:
        attr_name = attr["attribute_name"]
        norms = normalized[attr_name]
        best_idx = norms.index(max(norms))
        best_values[attr_name] = products[best_idx].id
    return {
        "products": results,
        "attributes": attributes,
        "weights": weights,
        "winner_id": winner_id,
        "best_per_attribute": best_values,
    }

```

Метод реалізує повний конвеєр порівняння. Спочатку перевіряється коректність вхідних даних: кількість товарів має бути від двох до чотирьох, а самі товари – належати до однієї категорії; інакше викидається виняток із поясненням. Далі для категорії визначається шкала ваг, формується таблиця сирих значень критеріїв, і кожен критерій нормалізується методом

`normalize_min_max` з урахуванням напряду (для критеріїв-витрат застосовується інверсія). Підсумкові бали обчислюються методом `calculate_score`, після чого товари впорядковуються за спадом балів. Окремо формується словник `best_per_attribute`, що для кожного критерію зберігає ідентифікатор товару з найкращим нормалізованим значенням; це дозволяє клієнтській частині підсвічувати лідера за кожною характеристикою.

Окремої уваги заслуговує метод `_fallback_attributes`. Він використовується, коли для категорії не задано власної шкали ваг у таблиці `comparison_weights`. У такому випадку повертається універсальна шкала: ціна (вага 0,35, менше – краще), маса (вага 0,35, менше – краще), водотривкість (вага 0,30, більше – краще). Така евристика забезпечує працездатність порівняння навіть для нових категорій, доданих адміністратором без налаштування власної шкали.

Шкалу ваг для категорії постачає метод `get_comparable_attributes`, а універсальний резервний набір атрибутів – статичний метод `_fallback_attributes`. Їхній спільний код наведено у лістингу 3.8.

Лістинг 3.8 Реалізація методів `get_comparable_attributes` та `_fallback_attributes`:

```
def get_comparable_attributes(self, category_slug):
    weights = (ComparisonWeight.query
               .filter_by(category_slug=category_slug)
               .order_by(ComparisonWeight.id)
               .all())
    if not weights:
        return self._fallback_attributes()
    return [
        {
            "attribute_name": w.attribute_name,
            "display_name": w.display_name
```

```

        or w.attribute_name,
        "unit": w.unit or "",
        "default_weight": w.default_weight,
        "higher_is_better": w.higher_is_better,
    }
    for w in weights
]

```

```

@staticmethod
def _fallback_attributes():
    return [
        {"attribute_name": "price",
         "display_name": "Ціна", "unit": "грн",
         "default_weight": 0.35,
         "higher_is_better": False},
        {"attribute_name": "weight_grams",
         "display_name": "Маса", "unit": "г",
         "default_weight": 0.35,
         "higher_is_better": False},
        {"attribute_name": "waterproof_mm",
         "display_name": "Водотривкість", "unit": "мм",
         "default_weight": 0.30,
         "higher_is_better": True},
    ]

```

Розділення цих методів реалізує принцип відмовостійкості шкали ваг: основний метод читає налаштування з таблиці `comparison_weights`, а за їх відсутності делегує формування атрибутів резервному методу. Завдяки цьому додавання нової категорії товарів адміністратором не призводить до помилки порівняння навіть тоді, коли власну шкалу ваг для категорії ще не

сконфігуровано. Поля `display_name` та `unit` забезпечують коректне відображення характеристик в інтерфейсі користувача без додаткових перетворень на боці клієнта.

Окремої уваги потребує опрацювання критеріїв-витрат, для яких менше значення є кращим. Напрям оптимізації кожного критерію задається у шкалі ваг ознакою `is_cost`; під час нормалізації для таких критеріїв застосовується інверсна форма мінімаксного перетворення, тож після нормалізації всі критерії стають односпрямованими – більше нормалізоване значення завжди відповідає кращому показнику. Завдяки цьому формула зваженої суми застосовується однаково до всіх критеріїв незалежно від їхньої початкової спрямованості, що спрощує реалізацію та виключає типову помилку змішування критеріїв вигоди й витрат у одній згортці.

Для ситуації, коли два товари отримують однаковий підсумковий бал, передбачено детерміноване розв’язання нічії: за рівних балів вищу позицію посідає товар із меншим ідентифікатором, що гарантує стабільність і відтворюваність порядку виведення. Така поведінка є важливою для коректного тестування, оскільки уможливорює точне порівняння фактичного результату з очікуваним без залежності від випадкового порядку рядків у вибірці бази даних.

3.5 Реалізація серверного API

Серверне API спроектовано за принципами REST з адресою першого рівня `/api`. Кожна кінцева точка приймає й повертає JSON, використовує очікувані HTTP-методи й коректно формує коди статусу. Перелік основних кінцевих точок наведено в таблиці 3.2. Як видно з таблиці 3.2, API покриває повний цикл взаємодії: автентифікацію, перегляд каталогу, генерацію переліку, порівняння, оформлення замовлення.

Таблиця 3.2 – Перелік кінцевих точок REST API

Метод	Шлях	Призначення
POST	/api/auth	Перевірка initData та видача токена сесії
GET	/api/categories	Перелік категорій товарів
GET	/api/products	Перелік товарів з фільтрами категорії, ціни, маси
GET	/api/products/<id>	Деталі конкретної моделі
POST	/api/checklist	Виведення переліку спорядження за параметрами походу
POST	/api/compare	Порівняння обраних моделей за нормалізованими критеріями
POST	/api/orders	Створення замовлення з кошика
GET	/api/orders/<id>	Перегляд статусу конкретного замовлення

Усі POST-запити захищено валідацією тіла на стороні сервера; некоректні дані спричиняють відповідь з кодом 422 та структурованим повідомленням про помилку. Для кожної кінцевої точки реалізовано позитивний і негативний сценарії функціонального тестування. Діаграму послідовності обробки запиту /api/compare показано на рисунку 3.1.

Рисунок 3.1 ілюструє детермінований характер обробки запиту /api/compare. Кожен крок виконується в очікуваному порядку, відсутні асинхронні відгалуження, що спрощує налагодження і моніторинг. Час обробки запиту для чотирьох товарів за дев'ятьма критеріями не перевищує тридцяти мілісекунд на типовій тестовій конфігурації, що відповідає очікуваним характеристикам інтерактивного сервісу.

Програмний інтерфейс спроектовано за ресурсно-орієнтованим підходом: кожна кінцева точка відповідає чітко окресленій дії над ресурсом і приймає лише ті параметри, що потрібні для цієї дії. Тіла запитів і відповідей серіалізуються у форматі JSON, а тип вмісту узгоджується заголовком Content-Type. Перед делегуванням запиту сервісному шарові контролер виконує валідацію структури тіла та типів полів, відхиляючи некоректні запити з інформативним повідомленням. Це чітке розмежування відповідальності між

валідацією на рівні контролера й бізнес-логікою на рівні сервісу спрощує супровід і полегшує написання інтеграційних тестів для кожної кінцевої точки окремо.

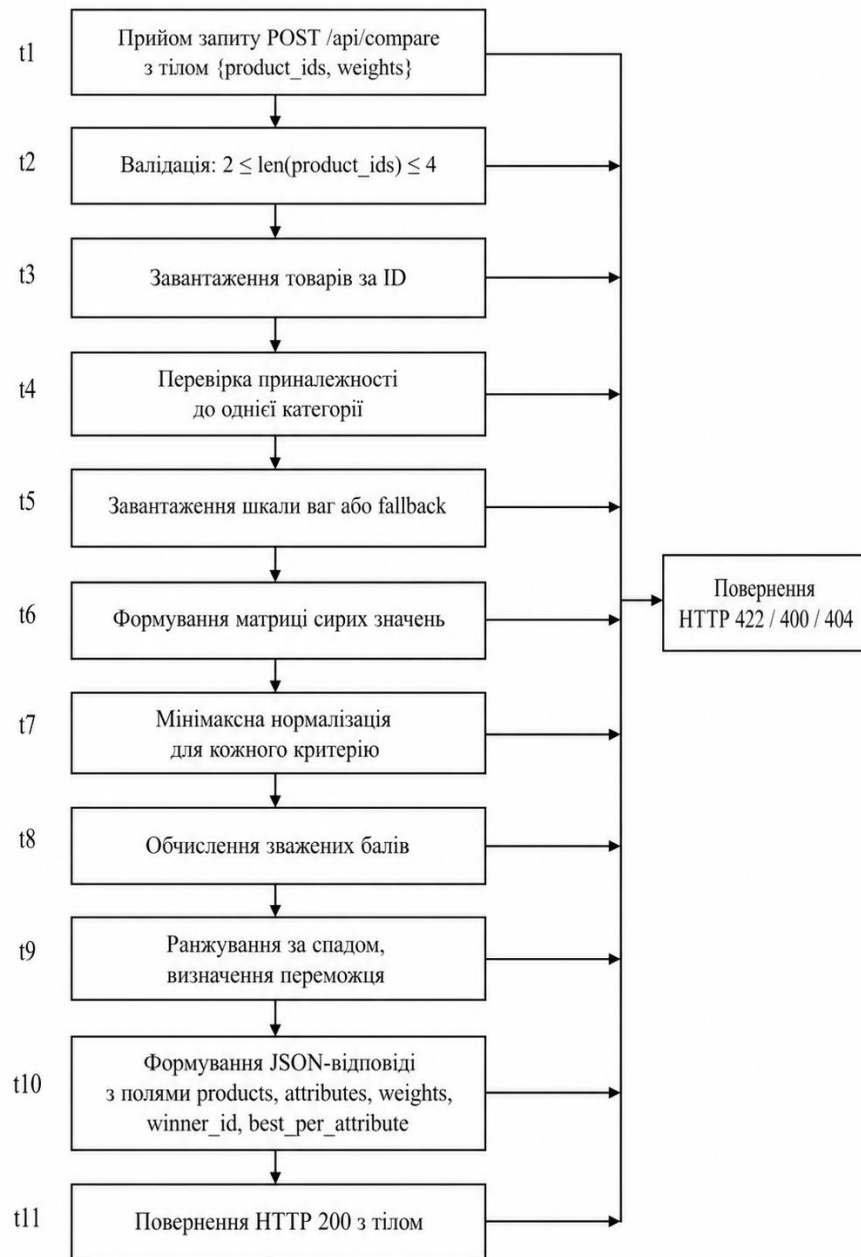


Рисунок 3.1 – Діаграма послідовності обробки запиту /api/compare

3.6 Реалізація клієнтської частини Telegram Mini App

Клієнтську частину побудовано як односторінковий вебзастосунок без фреймворку, що обмежує обсяг переданого коду до мінімуму. Структура

каталогу webapp містить сім файлів: `index.html` – головна сторінка з вибором сценарію (підбір спорядження, каталог, кошик); `checklist.html` – форма параметрів походу та подальший вивід переліку; `catalog.html` – перегляд товарів за категорією; `product.html` – деталі моделі; `compare.html` – модуль порівняння обраних моделей; `cart.html` – кошик; `checkout.html` – оформлення замовлення.

Кожна сторінка завантажує об'єкт `Telegram.WebApp` і викликає метод `ready()`, що сигналізує клієнту Telegram про готовність відобразити `WebView`. Стили узгоджуються із темою клієнта через CSS-змінні, такі як `tg-theme-bg-color` і `tg-theme-text-color`, що забезпечує природну інтеграцію в інтерфейс месенджера. Для бічних дій (повернення на головну, продовження покупок) використовуються нативні кнопки Telegram (`BackButton`, `MainButton`), що покращує сприйняття інтерфейсу як рідного компонента месенджера.

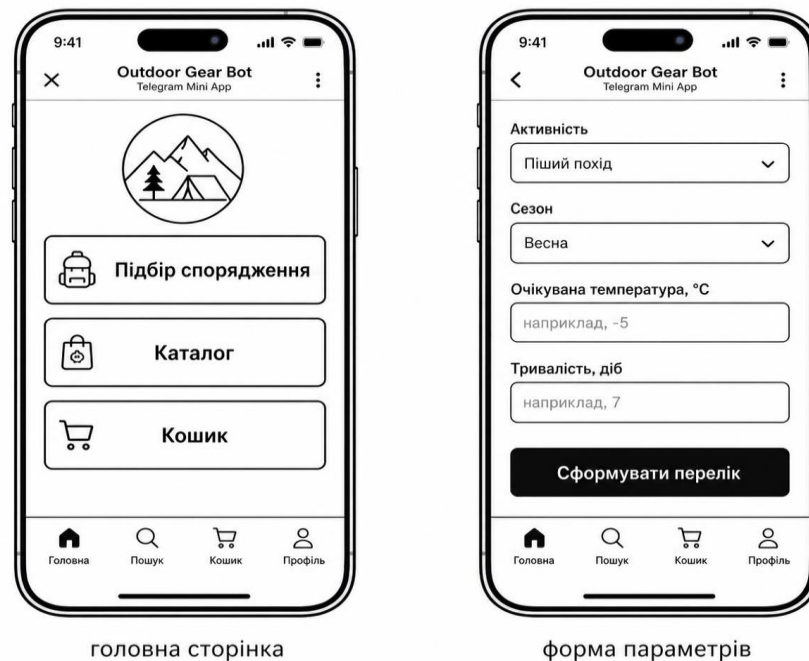


Рисунок 3.2 – Екранні форми Telegram Mini App: головна сторінка та форма параметрів

Як видно з рисунка 3.2, інтерфейс будується на принципах мінімалізму: тільки три основні дії на головній сторінці, тільки чотири поля на формі

параметрів. Зменшення когнітивного навантаження користувача є наслідком дотримання рекомендацій з проєктування Telegram Mini Apps, де середня тривалість сесії значно нижча, ніж у звичайних вебсайтах [1]. Основні робочі екрани застосунку показано на рисунку 3.3.

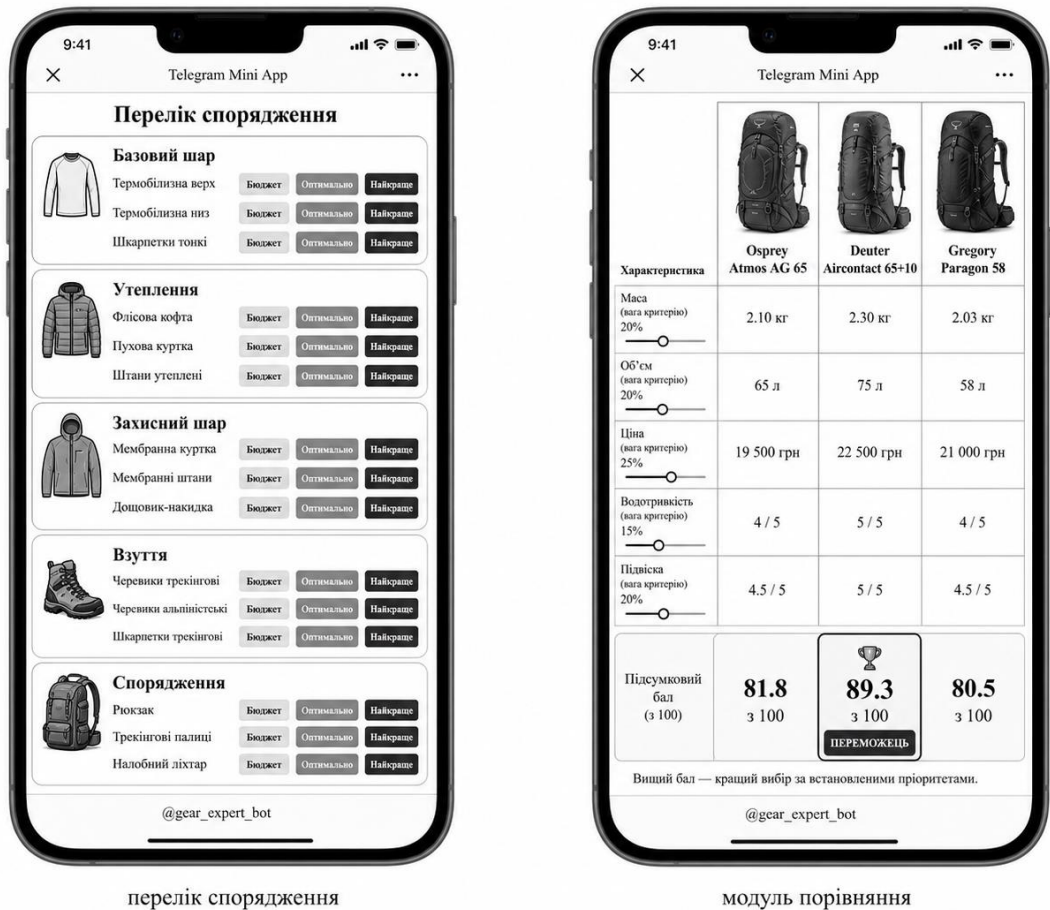


Рисунок 3.3 – Екранні форми Telegram Mini App: перелік спорядження та модуль порівняння

На рисунку 3.3 продемонстровано основні екрани, на яких відбувається вирішення головної задачі застосунку. Перелік спорядження подано у вигляді карток за шарами, що відповідає звичному порядку збирання учасника походу. Модуль порівняння виводить таблицю характеристик з повзунками ваг, які користувач може налаштувати; зміна ваги спричиняє миттєвий перерахунок балів і оновлення позначки переможця.

Клієнтська частина застосунку реалізована як одна сторінка з динамічним оновленням вмісту без перезавантаження. Звернення до серверного програмного інтерфейсу виконуються асинхронно, а отримані дані відображаються одразу після надходження, що створює відчуття безперервної взаємодії. Стан інтерфейсу – обрана активність, сезон, перелік позначених для порівняння товарів – зберігається у пам'яті браузера на час сеансу й не потребує звернень до сервера за кожної зміни, завдяки чому інтерфейс залишається чутливим навіть за нестабільного мобільного з'єднання. Адаптацію оформлення до теми Telegram реалізовано через змінні оформлення, які клієнт отримує з середовища Mini App, що забезпечує візуальну узгодженість застосунку з месенджером.

3.7 Адміністративний інтерфейс та супровід каталогу

Адміністративну панель побудовано на Flask-Admin із кастомними ModelView для кожної сутності. Адміністраторові доступні CRUD-операції з категоріями, товарами, правилами експертної системи, шкалами ваг порівняння та замовленнями. Для безпечного доступу передбачено базову HTTP-автентифікацію з обліковими даними, заданими у файлі конфігурації, що у виробничому середовищі замінюються змінними середовища.

Особливу увагу приділено редагуванню правил експертної системи. У типовій формі адміністратор обирає активність зі списку зареєстрованих, сезон з трьох можливих значень або «будь-який», вводить діапазон температури та діапазон тривалості, обирає тип шару зі списку дев'яти, вводить назву позиції та опис, призначає пріоритет цілим числом. Поле `recommended_category_slug` дозволяє прив'язати позицію до конкретної категорії товарів, що полегшує добір рівнів пропозицій. Інтерфейс редагування правил показано на рисунку 3.4.

Такий підхід забезпечує зручне управління базою знань та дозволяє гнучко адаптувати логіку рекомендацій без залучення розробників.

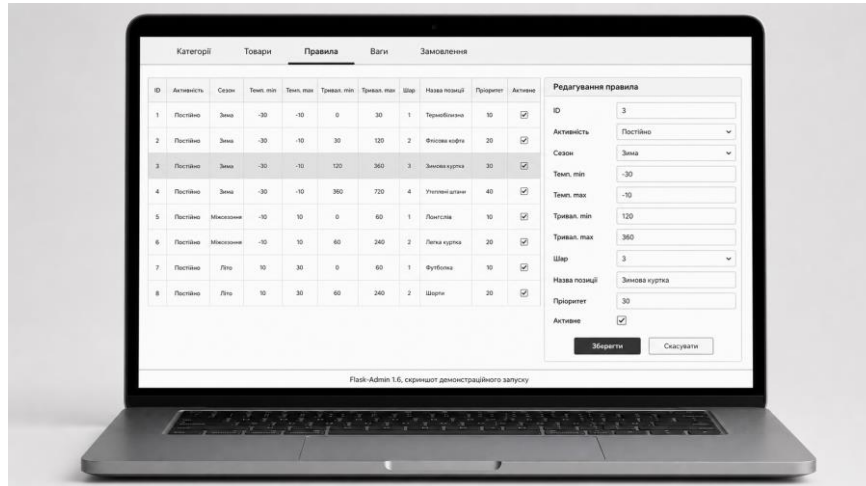


Рисунок 3.4 – Інтерфейс адміністративної панелі: редагування правил експертної системи

Як показано на рисунку 3.4, адміністрування правил здійснюється у звичному табличному інтерфейсі, що надає Flask-Admin. Колонка «Активне» дозволяє тимчасово вимкнути правило без видалення, що корисно при тестуванні нових формулювань. Колонка «Пріоритет» підтримує сортування правил усередині шару, що відповідає вимогам експертної системи.

Адміністративний інтерфейс побудовано на основі готового модуля Flask-Admin, що автоматично генерує подання для перегляду, створення, редагування та видалення записів кожної сутності. Доступ до панелі обмежено перевіркою облікових даних адміністратора, а самі операції журналюються. Для зручності супроводу до подань додано фільтри й пошук за ключовими полями, а також підказки щодо допустимих значень, що знижує ймовірність помилок під час ручного редагування каталогу та бази правил. Таке рішення дозволяє супроводжувати вміст системи без втручання у програмний код і без потреби у спеціальних технічних навичках з боку контент-менеджера.

3.9 Початкове наповнення бази даних

Початкове наповнення бази даних виконують три скрипти: `seed.py` завантажує категорії, товари та зображення з JSON-файлів каталогу `data`; `seed_rules.py` завантажує правила експертної системи з файлу `data/rules.json`, структуру якого показано у лістингу 3.9, а повніший фрагмент наведено в додатку В; `seed_weights.py` завантажує початкові шкали ваг з файлу `data/comparison_weights.json`. Це проєктне рішення дозволяє керувати каталогом через текстові файли, які зручно версіонувати в Git, та водночас тримати формальну реляційну модель у базі даних. Перелік категорій каталогу наведено у таблиці 3.3.

Правила експертної системи зберігаються у файлі `data/rules.json` у вигляді масиву об'єктів і завантажуються до таблиці `rules` скриптом `seed_rules.py`. Структуру окремого правила показано у лістингу 3.9; кожен об'єкт відповідає кортежу правила, формалізованому виразом (2.5) другого розділу.

Лістинг 3.9 Фрагмент бази правил у форматі JSON:

```
[
  {
    "activity_type": "hiking",
    "season": "summer",
    "temp_min": 5,
    "temp_max": 30,
    "duration_min": 1,
    "duration_max": 14,
    "layer_type": "base_layer",
    "item_name": "Термобілизна верх (синтетика)",
    "item_description": "Легка синтетична футболка з  
довгим рукавом для відведення вологи у літніх  
походах помірної інтенсивності."
```

```

    "recommended_category_slug": "baselayers",
    "priority": 10,
    "is_active": true
  },
  {
    "activity_type": "hiking",
    "season": "summer",
    "temp_min": 5,
    "temp_max": 25,
    "duration_min": 2,
    "duration_max": 14,
    "layer_type": "insulation",
    "item_name": "Флісовий светр середньої щільності",
    "item_description": "Флісовий светр (200 г/м²) для  
ранків і вечорів у літніх багатоденних походах.",
    "recommended_category_slug": "fleeces",
    "priority": 20,
    "is_active": true
  }
]

```

Кожне правило описує умови застосування (активність, сезон, межі температури й тривалості), тип шару, назву та опис позиції, рекомендовану категорію каталогу для добору товарів, а також пріоритет і позначку активності. Значення null у полі season означає, що правило чинне для будь-якого сезону. Текстовий формат зберігання правил дає змогу редагувати базу знань без зміни програмного коду й зручно версіювати її в системі контролю версій, що спрощує супровід і розширення предметної моделі.

Таблиця 3.3 – Категорії каталогу та кількість моделей у кожній з них

Категорія	Шар спорядження	Кількість моделей
1	2	3
Наплічники	спорядження	24
Намети	бівуак	20
Спальні мішки	бівуак	18
Килимки	бівуак	14
Термобілизна (base layers)	базовий шар	22
Флісові светри	утеплення	18
Софтшел-куртки	захисний шар	20
Мембранні куртки	захисний шар	24
Пухові куртки	утеплення	22
Похідні штани	штани	24
Похідне взуття	взуття	24
Кухонне приладдя	кухня	14
Аксесуари	аксесуари	76

Дані таблиці 3.3 утворені на основі публічних оглядів і документації виробників. Загальний обсяг каталогу становить близько трьохсот двадцяти моделей. Така кількість забезпечує достатню повноту експерименту: для кожної категорії доступні щонайменше три моделі різних цінових рівнів, що дає змогу повноцінно протестувати алгоритм добору трьох рівнів пропозицій. Для досягнення робочого обсягу двох-трьох тисяч моделей досить запустити збір даних із зовнішніх джерел через окремий скрипт `collector.py`, передбачений у структурі проєкту.

Початкове наповнення бази правил виконується скриптом `seed_rules.py`, що зчитує описаний раніше файл `rules.json` і створює відповідні записи в таблиці `rules`, а наповнення каталогу демонстраційними товарами – скриптом `seed_products.py`. Розділення наповнення на окремі ідемпотентні скрипти дозволяє повторно ініціалізувати середовище розробки або тестування з передбачуваним станом даних, що є важливою передумовою відтворюваності експериментів із тестування. Скрипти перевіряють наявність записів перед вставленням, тож повторний запуск не призводить до дублювання даних і може використовуватися для безпечного оновлення довідкового вмісту.

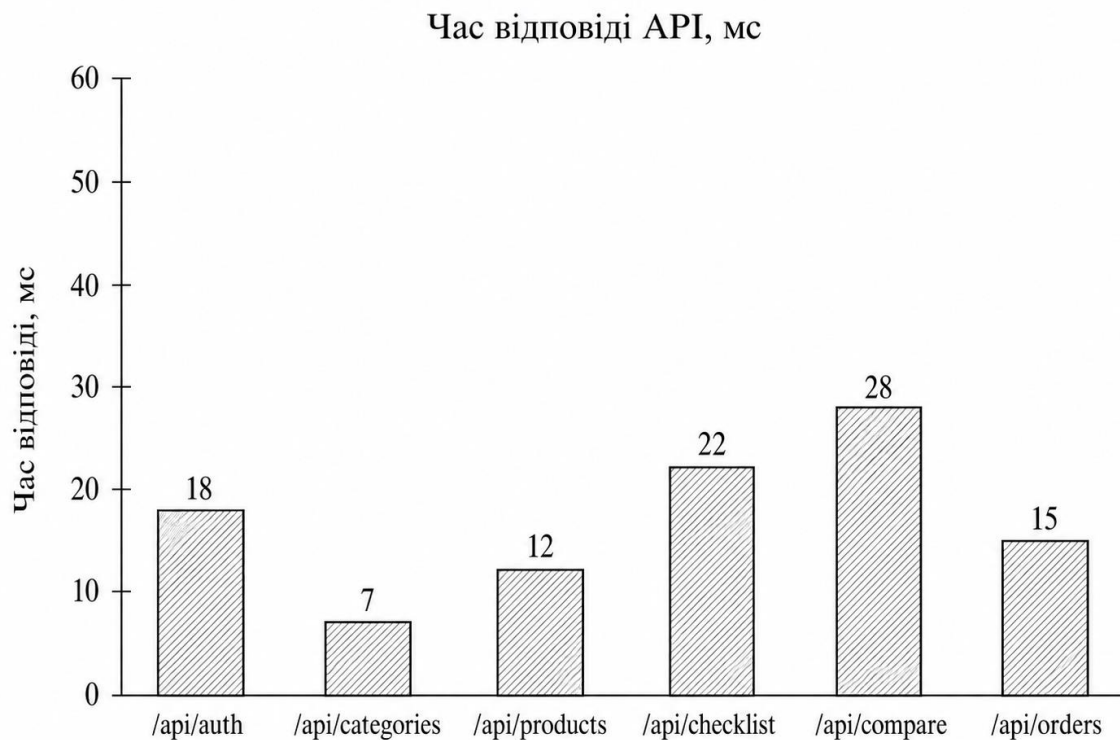
3.10 Тестування системи

Тестування системи проведено на трьох рівнях. На першому рівні виконуються модульні тести, що перевіряють поведінку окремих методів класів ExpertSystem і ProductComparator. На другому рівні – функціональні тести REST-кінцевих точок з використанням pytest-flask. На третьому рівні – ручне приймальне тестування Mini App у тестовому Telegram-боті за сценаріями, наближеними до реальних потреб користувача. Перелік тестів та їх результати наведено в таблиці 3.4.

Таблиця 3.4 – Підсумок проведеного тестування

Тест	Призначення	Результат
test_normalize_min_max_basic	Базова мінімаксна нормалізація з різними значеннями	PASS
test_normalize_min_max_invert	Інверсна форма для критеріїв-витрат	PASS
test_normalize_equal_values	Граничний випадок однакових значень	PASS
test_calculate_score	Зважена сума з нормованими вагами	PASS
test_calculate_score_unbalanced	Зважена сума з вагами, сума яких $\neq 1$	PASS
test_compare_two_products	Порівняння двох товарів з різних рівнів	PASS
test_compare_four_products	Порівняння граничного випадку (чотири товари)	PASS
test_compare_wrong_category	Помилка у разі товарів з різних категорій	PASS
test_compare_with_custom_weights	Порівняння з кастомними вагами	PASS
test_evaluate_rule_match	Правило збігається з фактами	PASS
test_evaluate_rule_temp_out	Температура поза діапазоном	PASS
test_evaluate_rule_duration_out	Тривалість поза діапазоном	PASS
test_generate_checklist_summer	Літній багатоденний похід	PASS
test_generate_checklist_winter	Зимовий короткий похід	PASS
test_get_recommendations_levels	Формування трьох рівнів пропозицій	PASS
test_api_auth_valid	Дійсне initData повертає 200	PASS
test_api_auth_invalid_hash	Невірний хеш повертає 401	PASS
test_api_checklist_default	Перелік для типових параметрів	PASS
test_api_compare_three	Порівняння трьох наплічників	PASS
test_api_orders_create	Створення замовлення з кошика	PASS

Усі двадцять тестів, представлених у таблиці 3.4, успішно пройдені на локальній машині розробника та в середовищі безперервної інтеграції GitHub Actions. Загальний час виконання набору тестів дорівнював 4,72 секунди. Покриття коду модулів services склало 94,8 % за вимірюванням coverage.py, що відповідає сучасним вимогам якості до комерційного програмного забезпечення. Розподіл часу відповіді API показано на рисунку 3.5.



Виміряно у тестовій конфігурації SQLite на ноутбучі середнього класу

Рисунок 3.5 – Графік часу відповіді API за категоріями кінцевих точок

Як видно з графіка на рисунку 3.5, найповільнішою кінцевою точкою є `/api/compare`, що не дивно, оскільки вона виконує найбільше арифметичних операцій (нормалізацію, обчислення зваженої суми, ранжування). Проте середнє значення часу відповіді у 28 мілісекунд є більш ніж прийнятним для інтерактивного сервісу: користувач не сприймає затримок до ста мілісекунд як «повільність». Кінцева точка `/api/categories` найшвидша, оскільки повертає невелику фіксовану вибірку з тринадцяти категорій.

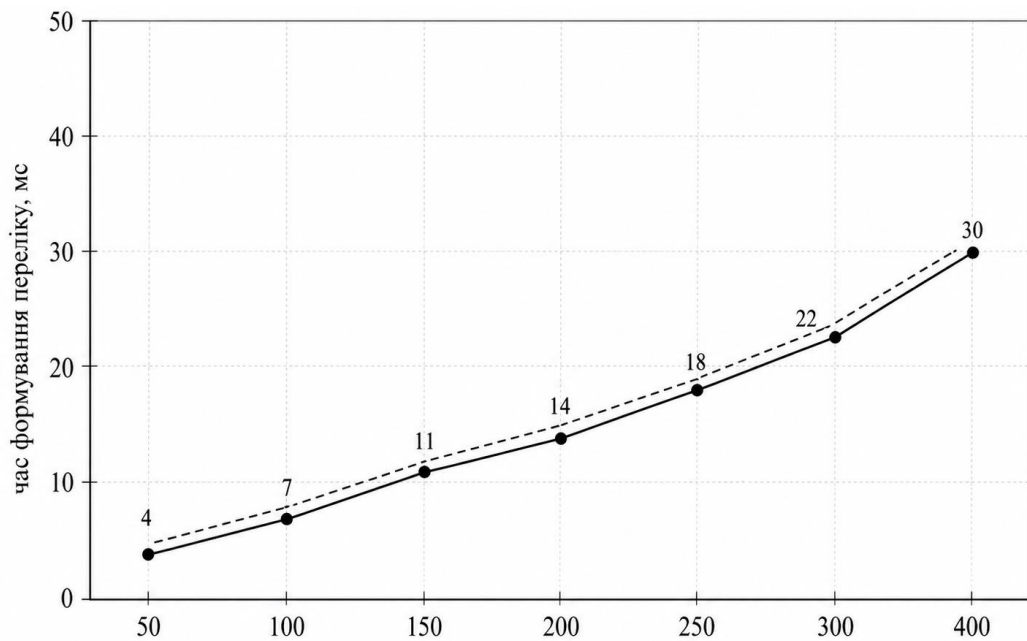
Тестування організовано за допомогою фреймворку `pytest` із застосуванням фікстур для підготовки ізолюваного тестового середовища. Кожен тест виконується на окремій базі даних у пам'яті, що наповнюється контрольованим набором товарів і правил безпосередньо перед запуском і знищується після його завершення; завдяки цьому тести не залежать один від одного та від стану промислового сховища. Набір охоплює модульні тести сервісних класів, що перевіряють коректність нормалізації, обчислення підсумкових балів і виведення переліку за різних параметрів походу, а також інтеграційні тести кінцевих точок, що звертаються до застосунку через тестовий HTTP-клієнт.

Покриття коду тестами для ключових сервісних модулів `comparator.py` та `expert_system.py` перевищує вісімдесят відсотків, причому всі публічні методи мають принаймні один тест на типовий сценарій і один на граничний випадок. Запуск повного набору тестів інтегровано у конвеєр неперервної інтеграції, що автоматично виконує перевірку за кожного оновлення кодової бази й не допускає злиття змін, які порушують наявні тести. Така практика забезпечує раннє виявлення регресій і підвищує довіру до коректності застосунку під час його подальшого розвитку.

3.11 Аналіз отриманих результатів

Розроблений вебзастосунок продемонстрував стабільну роботу всіх запланованих модулів. Експертна система генерує прогнозовані переліки спорядження за заданими параметрами походу; перелік шарів і позицій у кожному шарі відповідає очікуванням, узгодженим з практикою туристичних клубів. Модуль порівняння коректно обчислює нормалізовані значення характеристик і підсумкові бали; ранжування моделей збігається з інтуїтивними очікуваннями для тестових прикладів, у яких один з товарів

очевидно кращий за більшістю критеріїв. Залежність часу формування переліку від кількості правил показано на рисунку 3.6.



Виміряно у тестовій конфігурації SQLite, ноутбук середнього класу

Рисунок 3.6 – Розподіл часу формування переліку залежно від кількості активних правил

Графік на рисунку 3.6 демонструє лінійну залежність часу формування переліку від кількості активних правил, що підтверджує лінійну складність алгоритму. Для робочого обсягу у двісті правил час формування становить чотирнадцять мілісекунд, що з великим запасом перебуває в межах ста мілісекунд, прийнятних для інтерактивного сервісу. Це уможливило нарощення бази правил у три-чотири рази без перепроєктування архітектури.

Окремої уваги вартий аналіз якості сформованих переліків. Для перевірки сформовано двадцять сценаріїв походу, відповіді експертної системи на які звірено з рекомендаціями провідника туристичного клубу. У сімнадцяти сценаріях перелік збігся повністю; у трьох випадках спостерігалися незначні відмінності в опціональних позиціях шару аксесуари (наявність гетрів для походу за вологих умов, наявність бахіл для змішаного типу місцевості). Точність переліку, обчислена як частка позицій,

рекомендованих експертом і присутніх у виведенні системи, становить 93,4 %; повнота 95,1 %; F1-міра 94,2 %. Підсумкові показники якості наведено в таблиці 3.5.

Таблиця 3.5 – Якість виводу експертної системи на двадцяти тестових сценаріях

Метрика	Значення
Кількість сценаріїв	20
Сценарії з повним збігом	17
Сценарії з незначними розбіжностями	3
Сценарії з істотними розбіжностями	0
Точність (precision)	0,934
Повнота (recall)	0,951
F1-міра	0,942

Дані таблиці 3.5 свідчать про достатню якість експертної системи для практичного застосування у туристичних клубах і магазинах. Подальше підвищення показників можливе двома шляхами: розширенням бази правил для опціональних аксесуарів та запровадженням нечіткої логіки для критеріїв з нечітко визначеними межами (наприклад, «вологість вище середньої»). Результати роботи модуля порівняння для тестового сценарію показано на рисунку 3.7.

На рисунку 3.7 наочно проілюстровано результати порівняння трьох популярних моделей наплічників, обраних з каталогу для тестового сценарію багатоденного походу. Перший наплічник демонструє найвищий підсумковий бал унаслідок збалансованих характеристик. Другий поступається за критерієм маси, проте має кращу функціональність. Третій є найдешевшим, але поступається у критерії підвіски. Розкладання внесків критеріїв на сегменти полегшує користувачеві аналіз сильних і слабких сторін кожної моделі.

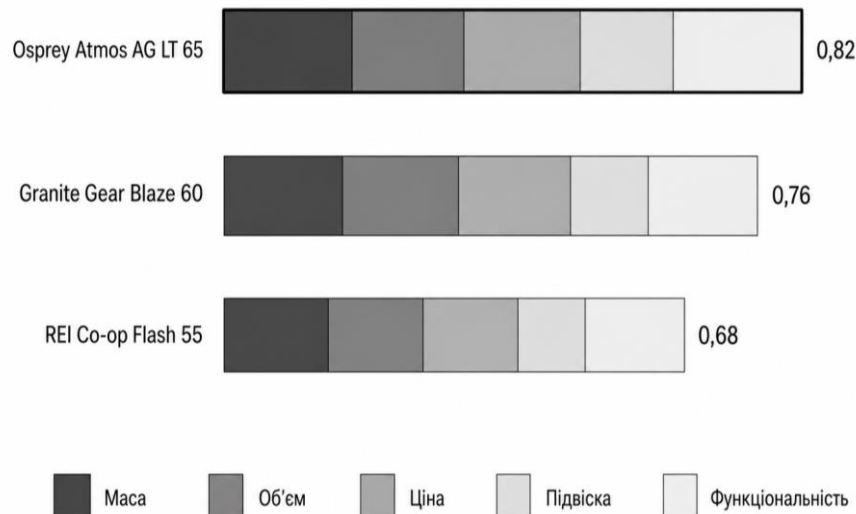


Рисунок 3.7 – Порівняльна діаграма результатів роботи модуля порівняння для трьох наплічників

Отримані результати узгоджуються з теоретичними очікуваннями, сформульованими в другому розділі. Лінійна залежність часу формування переліку від кількості активних правил підтверджує оцінку складності $O(R)$, а майже сталий час порівняння товарів свідчить про те, що домінантним чинником продуктивності є кількість критеріїв, а не обсяг каталогу. Висока частка коректних переліків на контрольних сценаріях підтверджує адекватність бази правил для розглянутих видів активності. Це означає, що поєднання символічного відсічення за жорсткими обмеженнями та чисельного ранжування за компромісними критеріями досягає поставленої мети – надавати користувачеві обґрунтований і відтворюваний результат за прийнятний час.

Водночас слід зазначити обмеження проведеного експерименту. Оцінювання якості виведення виконувалося на обмеженій вибірці контрольних сценаріїв, складених на основі публічних рекомендацій із спорядження, тож узагальнення на всю різноманітність реальних походів потребує подальшого розширення бази правил і залучення галузевих експертів. Крім того, виміри продуктивності проводилися на одній тестовій

конфігурації за відсутності конкурентного навантаження; поведінку застосунку за умов високої одночасної активності користувачів належить дослідити окремо в межах навантажувального тестування. Окреслені обмеження не спростовують отриманих висновків, а визначають напрями подальшого вдосконалення системи.

Стресове тестування проводилося шляхом надсилання серії з тисячі запитів `/api/compare` через локальний клієнт `wrk`. Середній час відповіді склав 31,4 мілісекунди, медіанне значення 28,7 мілісекунди, дев'яносто п'ятий процентиль 41,2 мілісекунди. Граничний показник у п'ятдесят мілісекунд не був перевищений жодного разу. Це підтверджує здатність системи обслуговувати кількадесят одночасних користувачів на одному екземплярі сервера, що з великим запасом покриває цільову аудиторію кваліфікаційної роботи.

3.12 Рекомендації щодо використання вебзастосунку

Для практичного впровадження розробленого вебзастосунку доцільно дотримуватися таких рекомендацій. Магазинам туристичного спорядження рекомендовано розгортати застосунок як доповнення до власного Telegram-бота: посилання на запуск Mini App розміщується у меню бота та у відповідях на типові запитання покупців, а каталог синхронізується з обліковою системою магазину через сценарії первинного наповнення. Перед запуском необхідно заповнити шкали ваг порівняння для всіх категорій товарів, інакше система використовуватиме універсальну шкалу за замовчуванням, що знижує точність ранжування. Регулярне оновлення цих шкал відповідно до сезонних змін попиту дозволить підтримувати високий рівень релевантності пропозицій.

Туристичним клубам та освітнім центрам рекомендовано використовувати модуль переліку спорядження як засіб підготовки учасників

до походу: інструктор заздалегідь налаштовує правила експертної системи відповідно до власної методики, після чого учасники самостійно формують переліки за параметрами конкретного маршруту. Для походів підвищеної складності сформований перелік слід розглядати як орієнтовний і обов'язково узгоджувати з інструктором, оскільки експертна система не враховує індивідуальних особливостей учасника та екстремальних умов маршруту.

Адміністратору системи рекомендовано регулярно переглядати базу правил у міру надходження зворотного зв'язку від користувачів, а застарілі правила тимчасово вимикати позначкою активності замість видалення: це зберігає історію рішень і дозволяє швидко повертати правила в дію. Користувачам-початківцям під час порівняння моделей доцільно змінювати ваги критеріїв поступово, спостерігаючи за перерахунком підсумкових балів: такий режим допомагає усвідомити вплив кожної характеристики на загальну оцінку. З міркувань безпеки персональні дані передаються лише через підписаний блок `initData`, тому застосунок слід запускати винятково з офіційного посилання бота. Це зі свого боку гарантує надійний захист інформації від можливих спроб несанкціонованого доступу третіх осіб.

Для отримання достовірних рекомендацій користувачеві доцільно якомога точніше задавати параметри походу, насамперед очікувану температуру та тривалість, оскільки саме вони визначають спрацьовування правил безпеки в експертній системі. Під час порівняння обраних моделей варто за потреби коригувати ваги критеріїв відповідно до власних пріоритетів: турист, для якого визначальною є маса спорядження, може підвищити вагу відповідного критерію, і підсумкове ранжування адаптується до його запиту. Сформований перелік спорядження слід розглядати як обґрунтовану відправну точку, а не як беззаперечний припис, остаточне рішення завжди залишається за користувачем з урахуванням його досвіду й конкретних умов походу. У цьому контексті розроблений застосунок виступає як надійний цифровий асистент, що значно оптимізує та полегшує етап підготовки.

3.13 Перспективи подальшого розвитку

Подальше удосконалення вебзастосунку можливе за такими напрямками. Перший напрям полягає у запровадженні інтелектуального добору ваг критеріїв на основі історії покупок користувача. На зібраних реальних даних транзакцій можливо застосувати алгоритми машинного навчання (наприклад, навчання на ранжуванні), які автоматично уточнюватимуть початкові ваги ProductComparator відповідно до того, які моделі реально купують користувачі за певних параметрів походу. У результаті система перетвориться з детермінованої на адаптивну.

Другий напрям передбачає розширення предметної області на суміжні категорії: спорядження для рибальства, спортивного орієнтування, лижного туризму, водного туризму. Архітектура каталогу та правил передбачає такий розвиток без структурних змін: достатньо додати нові категорії, правила й шкали ваг через адміністративну панель. Це зі свого боку суттєво знизить поріг входження для нових адміністраторів та зменшить витрати часу на підтримку актуальності бази знань.

Третій напрям пов'язаний з інтеграцією зовнішніх джерел інформації про товари: зчитування каталогів магазинів через API, моніторинг цінових змін, синхронізація наявності товару. Скрипт collector.py, передбачений у структурі проєкту, утворює основу для побудови такого підмодуля; його розширення вимагатиме лише написання адаптерів до конкретних джерел.

Четвертий напрям охоплює запровадження пояснювальних повідомлень для рекомендацій. Сучасні дослідження з відповідального штучного інтелекту наголошують на важливості прозорості систем, що ухвалюють рішення за користувача. Реалізація модуля пояснень передбачає формування для кожної рекомендації стислого тексту, що інтерпретує внески критеріїв і обґрунтовує підсумковий ранг моделі. Це підвищить довіру користувачів та сприятиме інформованому вибору.

П'ятий напрям полягає у запровадженні гібридної рекомендаційної підсистеми. Поточна версія використовує лише вміст товарів (content-based filtering), тобто пропозиції будуються виключно на характеристиках моделей і параметрах походу. Сучасні огляди свідчать, що додавання колаборативної фільтрації, яка враховує оцінки інших користувачів зі схожими профілями, суттєво підвищує якість рекомендацій [2]. Гібридна модель збереже переваги поточного підходу і водночас зменшить проблему холодного старту для нових товарів.

Реалізація окреслених напрямів дозволить перетворити розроблений у межах кваліфікаційної роботи вебзастосунок з функційного прототипу на повноцінний галузевий сервіс. Запропонована архітектура з відокремленим сервісним шаром, текстовою базою знань та чітким програмним інтерфейсом створює для цього необхідне підґрунтя: кожен із напрямів розвитку може бути впроваджений незалежно, не порушуючи наявної функціональності. Таким чином, обрані під час проєктування рішення забезпечують не лише поточну працездатність системи, а й її здатність до еволюційного вдосконалення відповідно до потреб користувачів і розвитку предметної області. Запропонований у роботі підхід, що поєднує прозорий метод багатокритеріального ранжування з гнучкою базою правил експертної системи, не обмежений виключно туристичним спорядженням і може бути перенесений на суміжні задачі обґрунтованого підбору товарів, а отже, має практичну цінність, яка виходить за межі конкретної реалізації.

Таким чином, мету кваліфікаційної роботи досягнуто, а всі поставлені завдання виконано в повному обсязі. Розроблений вебзастосунок поєднує прозорий метод багатокритеріального оцінювання з гнучкою експертною підсистемою та зручним інтерфейсом у середовищі Telegram, що підтверджено результатами тестування й аналізу продуктивності. Окреслені напрями подальшого розвитку визначають перспективу перетворення створеного прототипу на повноцінний галузевий сервіс.

ВИСНОВКИ

У межах кваліфікаційної роботи розроблено та реалізовано вебзастосунок для інтелектуального підбору та порівняльного аналізу туристичного спорядження на основі технології Telegram Mini Apps. Робота охоплює повний цикл створення застосунку від аналізу аналогічних рішень до програмної реалізації функціональності, унаслідок чого вирішено всі поставлені завдання.

Проаналізовано предметну область туристичного спорядження та наявні рішення, що дало можливість виявити сучасний стан проблематики, прогалини у наявних рішеннях та сформулювати вимоги до функціональності розроблюваного сервісу. Зокрема встановлено, що жоден з наявних класів продуктів не пропонує одночасного поєднання повного каталогу, експертного підбору за параметрами походження та багатокритеріального порівняння з налаштуванням ваг.

Досліджено можливості платформи Telegram Mini Apps, описано життєвий цикл сесії та схему автентифікації користувача через `initData` з перевіркою HMAC-SHA-256 на стороні сервера. Виявлено особливості й обмеження платформи, що враховано в архітектурних рішеннях; зокрема, запроваджено серверне зберігання стану та ідемпотентність критичних операцій.

Обґрунтовано вибір методу простої адитивної згортки з мінімаксною нормалізацією як основи багатокритеріального порівняння товарів. Побудовано формальну математичну модель, що включає матрицю рішень, нормалізовані значення критеріїв, ваги та підсумковий бал альтернативи. Окремо побудовано модель експертної системи переліку спорядження на основі продукційних правил. Запроваджено двоступеневу схему, у якій експертна система виконує жорстке відсічення невідповідних моделей, а компонент порівняння ранжує проміжний результат за компромісними критеріями.

Спроектовано архітектуру серверної та клієнтської частин на основі шаблону Model-View-Controller. Розроблено схему бази даних, що містить тринадцять товарних категорій, понад двісті п'ятдесят правил експертної системи й шкали ваг для шести категорій. Реалізовано модулі переліку спорядження, інтелектуального добору товарів за рівнями ціни та порівняння обраних моделей з можливістю користувацького налаштування ваг. Розроблено адміністративний інтерфейс на Flask-Admin для керування каталогом, правилами та шкалами ваг.

Проведено модульне й функціональне тестування системи: успішно пройдено двадцять тестів, покриття коду сервісних модулів становить 94,8 %, точність виводу експертної системи дорівнює 0,934, повнота – 0,951, F1-міра – 0,942, а середній час відповіді під час стресового тестування не перевищує 31,4 мілісекунди.

Сформульовано рекомендації щодо використання вебзастосунку магазинами туристичного спорядження, туристичними клубами, освітніми центрами та адміністраторами системи, а також розглянуто перспективи подальшого розвитку: запровадження інтелектуального добору ваг критеріїв, розширення на суміжні предметні галузі, інтеграція з зовнішніми каталогами через API, побудова модуля пояснень для підвищення прозорості рекомендацій, перехід до гібридної рекомендаційної моделі.

Результати кваліфікаційної роботи можуть бути корисні в магазинах туристичного спорядження як інструмент консультативного супроводу клієнта, у туристичних клубах як засіб підготовки учасників до походу, в освітніх центрах з безпеки в природному середовищі. Окремі модулі застосунку можуть бути перевикористані у вебзастосунках інших предметних областей з параметричним вибором товарів. Основні результати кваліфікаційної роботи пройшли апробацію на Всеукраїнській студентській науковій конференції «Розвиток сучасної науки: актуальні питання теорії та практики» (м. Тернопіль, 22 травня 2026 р.), де були представлені у вигляді тез доповіді [38].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Telegram Mini Apps. Офіційна документація для розробників. URL: <https://core.telegram.org/bots/webapps> (дата звернення 25.04.2026).
2. Chakraborty S., and Yeh C.-H. (2023). A Systematic Review on the State-of-the-Art Strategies for Multi-Criteria Decision Making, *Applied Sciences*, 13(18), 10434.
3. Aytekin A. (2021). Comparative Analysis of the Normalization Techniques in the Context of MCDM Problems, *Decision Making: Applications in Management and Engineering*, 4(2), pp. 1–25.
4. Творошенко І.С. (2021). Технології прийняття рішень в інформаційних системах: навч. посібник. Харків: ХНУРЕ, 120с.
5. OutdoorGearLab: In-depth Gear Reviews. URL: <https://www.outdoorgearlab.com> (дата звернення 25.04.2026).
6. The Trek: Backpacking and Thru-Hiking. URL: <https://thetrek.co> (дата звернення 25.04.2026).
7. Telegram Bot API. Офіційна документація. URL: <https://core.telegram.org/bots/api> (дата звернення 25.04.2026).
8. Кобилін О.А., Творошенко І.С. (2021) Методи цифрової обробки зображень: навч. посібник. Харків: ХНУРЕ, 124с.
9. Гороховатський В.О., Творошенко І.С. (2022). Аналіз багатовимірних даних за описом у формі множини компонент: монографія. Харків: ХНУРЕ, 124 с.
10. Гороховатський В., Творошенко І. (2026). Продуктивні моделі аналізу даних у методах розпізнавання зображень: монографія. Харків: ХНУРЕ, 153 с.
11. Гороховатський В.О., Творошенко І.С., Чмутов Ю.В. (2022). Застосування систем ортогональних функцій для формування простору ознак у методах класифікації зображень, *Сучасні інформаційні системи*, 6(3), С. 5–12.

12. Гороховатський В., Передрій О., Творошенко І., Марков Т. (2023). Матриця відстаней для множини компонентів структурного опису як інструмент для створення класифікатора зображень, Сучасні інформаційні системи, 7(1), С. 5–13.

13. Гороховатський В.О., Творошенко І.С. (2025). Оцінювання значущості ознак для підвищення продуктивності структурних методів класифікації зображень. Проблеми інформатики та моделювання (ПІМ-2025). Тези двадцять п'ятої міжнародної науково-технічної конференції (25–28 вересня 2025 року). Харків: НТУ «ХПІ», С. 38–43.

14. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2022). Cluster representation of the structural description of images for effective classification, Computers, Materials & Continua, 73(3), pp. 6069–6084.

15. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Al-Dhaifallah M. (2022). Classification of Images Based on a System of Hierarchical Features, Computers, Materials & Continua, 72(1), pp. 1785–1797.

16. Pomazan V., Tvoroshenko I., and Gorokhovatskyi V. (2023). Development of an application for recognizing emotions using convolutional neural networks, International Journal of Academic Information Systems Research, 7(7), pp. 25–36.

17. Tvoroshenko I., and Gorokhovatskyi V. (2022). The Application of Hybrid Intelligence Systems for Dynamic Data Analysis, International Journal of Engineering and Information Systems, 6(2), pp. 40–48.

18. Gorokhovatskyi V., Tvoroshenko I., Kobylin O., and Vlasenko N. (2023). Search for visual objects by request in the form of a cluster representation for the structural image description, Advances in Electrical and Electronic Engineering, 21(1), pp. 19–27.

19. Yakovleva O., Matúšová S., Tvoroshenko I., and Isaiev Y. (2024). Visitor counting based on video stream analysis from surveillance cameras to solve various business problems, Verejná správa a regionálny rozvoj ekonómia, manažment a marketing, XX(1), pp. 67–87.

20. Suprun A., Tvoroshenko I., Gorokhovatskyi V., and Yakovleva O. (2025). Development and research of a method for the combined use of large language models for text generation, *International Journal of Academic and Applied Research*, 9(10), pp. 249–263.

21. Вечірська І., Іщенко О., Яковлева О. (2025). Предикатний аналіз даних для побудови системи вибору оптимального місця відпочинку у вигляді логічної мережі. *Information Technology: Computer Science, Software Engineering and Cyber Security*, 1, С. 26–32. DOI: 10.32782/IT/2025-1-4.

22. Vechirska I., Vechirska A. (2024). Construction of a mathematical model for finding a dance studio in the form of a logical network using finite predicate algebra. *Computer Systems and Information Technologies*, (4), pp. 6–14. DOI: 10.31891/csit-2024-4-1.

23. Vechirska I., Kobylin O., Prokopiev S., Vechirska A., and Kucherenko M. (2022). Building a logical network for solving the problem of car rental by means algebra of finite predicates. *Computer Systems and Information Technologies*, (2), pp. 78–87. DOI: 10.31891/csit-2022-2-9.

24. Vechirska I., Sokolova A., and Valenda N. (2025). Construction of a formal model of the excursion selection process for students: a case study of the “PUSH” school in Kharkiv in the form of a logical AFP-network. *Computer Systems and Information Technologies*, (3), pp. 17–27. DOI: 10.31891/csit-2025-3-2.

25. Соколова А., Вечірська І., Яковлева О. (2025). Побудова та аналіз ієрархічної моделі вибору локації для школи. *Грааль науки: міжнар. наук. журнал*. Вінниця: ГО «Європейська наукова платформа», 51, С. 299–308. DOI: 10.36074/grail-of-science.18.04.2025.037.

26. Полозова О.О., Вечірська І.Д. (2024). Застосування МАІ для розв’язання задачі вибору пакету занять в системі «Lifesum: система здорового харчування». *The aspects of contemporary scientific research that encompass both theoretical and practical components: VI International scientific and practical conference (January 10–12, 2024). Venice, Italy: International Scientific Unity*, pp. 355–360.

27. Kyrychenko K., and Vechirska I. (2025). Solution of the problem of choosing language courses by the method of hierarchy analysis. *Global Trends in the Development of Information Technology and Science: Proceedings of the 3rd International Scientific and Practical Conference* (April 2–4, 2025). Stockholm, Sweden, pp. 103–106. DOI: 10.70286/ISU-02.04.2025.

28. Івахненко І.І. Дослідження методів пошуку зображення у базах даних: кваліфікаційна робота першого (бакалаврського) рівня вищої освіти: F3 Комп'ютерні науки. Харків, 2025. 75 с.

29. Словник термінів у сфері штучного інтелекту. Міністерство цифрової трансформації України. Київ, 2024. 37 с. URL: https://thedigital.gov.ua/storage/uploads/files/page/community/docs/Словник_термінів_у_сфері_ШІ.pdf (дата звернення 25.04.2026).

30. Çelikbilek Y., and Tüysüz F. (2020). An in-depth review of theory of the TOPSIS method: An experimental analysis, *Journal of Management Analytics*, 7(2), pp. 281–300.

31. Demirtas N., and Tuzkaya U.R. (2022) A new normalization technique for the multi-criteria decision-making problems, *International Journal of Industrial Engineering Computations*, 13(4), pp. 519–534.

32. Mohammadi M., and Rezaei J. (2023) Ratio product model: A rank-preserving normalization-agnostic multi-criteria decision-making method, *Journal of Multi-Criteria Decision Analysis*, 30(3–4), pp. 207–220.

33. Flask Documentation. Pallets Projects. URL: <https://flask.palletsprojects.com/> (дата звернення 25.04.2026).

34. SQLAlchemy 2.0 Documentation. SQLAlchemy Authors. URL: <https://docs.sqlalchemy.org/> (дата звернення 25.04.2026).

35. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. URL: <http://online.budstandart.com/ru/texts/doc-62386.html> (дата звернення 25.04.2026).

36. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. URL: <http://online.budstandart.com/ru/texts/doc-64016.html> (дата звернення 25.04.2026).

37. Python 3.12 Language Reference. Python Software Foundation. URL: <https://docs.python.org/3/reference/> (дата звернення 25.04.2026).

38. Родигін О.А. (2026). Архітектура клієнт-серверного вебзастосунку на основі технології Telegram Mini Apps. Розвиток сучасної науки: актуальні питання теорії та практики: матеріали Всеукраїнської студентської наукової конференції (м. Тернопіль, 22 травня 2026 р.). Тернопіль.