

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій  
Кафедра комп'ютерних систем та мереж

**Пояснювальна записка**  
до магістерської роботи

\_\_\_\_\_магістра

(ступінь вищої освіти)

на тему **КОМП'ЮТЕРНА СИСТЕМА МОДЕЛЮВАННЯ ЕФЕКТИВНОГО  
РОЗПОДІЛЕННЯ ВАГИ РАЦІОНУ МІЖ УЧАСНИКАМИ ТУРИСТИЧНОЇ ПОДОРОЖІ**

Виконав: студент 2 курсу, групи КНТ-513м  
спеціальності \_\_\_\_\_

123 Комп'ютерна інженерія

(код і найменування спеціальності)

Освітня програма (спеціалізація) \_\_\_\_\_

Комп'ютерні системи та мережі

\_\_\_\_\_БОРОВИК А. В.

(ПРИЗВИЩЕ та ініціали)

Керівник \_\_\_\_\_СКРУПСЬКИЙ С.Ю.

(ПРИЗВИЩЕ та ініціали)

Рецензент \_\_\_\_\_ЄФИМЕНКО М.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
**Національний університет «Запорізька політехніка»**

Факультет Комп'ютерних наук і технологій  
Кафедра «Комп'ютерні системи та мережі»  
Ступінь вищої освіти магістерський  
Спеціальність 123 Комп'ютерна інженерія  
(код і найменування)  
Освітня програма (спеціалізація) «Комп'ютерна системи та мережі»  
(назва освітньої програми (спеціалізації))

**ЗАТВЕРДЖУЮ**  
Зав. кафедри Кудерметов Р.К.  
“ ” 2024 року

**З А В Д А Н Н Я**  
**НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТА**

БОРОВИК Антон Володимирович  
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Комп'ютерна система моделювання поведінки  
людей в екстремальних ситуаціях  
керівник проєкту (роботи) к. т. н., доцент, СКРУПСЬКИЙ Степан Юрійович  
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом вищого навчального закладу від “18” жовтня 2024 року № 149

2. Строк подання студентом проєкту (роботи) 10 грудня 2024 року  
3. Вихідні дані до проєкту (роботи) Дані про учасників походу, алгоритмічні  
обмеження

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно  
розробити) 1) Аналіз існуючих рішень і технологій;  
2) Розробка Клієнт-серверної системи;  
3) Дослідження системи.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

## 6. Консультанти розділів проєкту (роботи)

| Розділ        | ПРИЗВИЩЕ, ініціали та посада консультанта | Підпис, дата   |                           |
|---------------|-------------------------------------------|----------------|---------------------------|
|               |                                           | завдання видав | прийняв виконане завдання |
| 1-3           | СКРУПСЬКИЙ С.Ю., доцент                   |                |                           |
| нормоконтроль | ЩЕРБАК Н.В., ст. викл.                    |                |                           |
|               |                                           |                |                           |
|               |                                           |                |                           |
|               |                                           |                |                           |
|               |                                           |                |                           |

7. Дата видачі завдання 01.10.2024 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів магістерської роботи                                                                                                                                               | Строк виконання етапів проєкту ( роботи ) | Примітка |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|----------|
| 1     | Аналіз наявних методів та алгоритмів існуючих рішень для розрахунку і розподілу ресурсів у туристичних походах, перегляд отриманих результатів, виділення недоліків та переваг. | 20.10.2024 р.                             |          |
| 2     | Встановлення та налаштування програмного забезпечення                                                                                                                           | 26.10.2024 р.                             |          |
| 3     | Аналіз технологій                                                                                                                                                               | 27.10.2024 р.                             |          |
| 4     | Розробка програмних компонентів системи                                                                                                                                         | 14.11.2024 р.                             |          |
| 5     | Дослідження розробленої системи                                                                                                                                                 | 23.11.2024 р.                             |          |
| 6     | Оформлення отриманих результатів у ПЗ                                                                                                                                           | 04.12.2024 р.                             |          |
| 7     | Оформлення графічного матеріалу                                                                                                                                                 | 07.12.2024 р.                             |          |
| 8     | Оформлення допоміжного матеріалу                                                                                                                                                | 10.12.2024 р.                             |          |
|       |                                                                                                                                                                                 |                                           |          |
|       |                                                                                                                                                                                 |                                           |          |
|       |                                                                                                                                                                                 |                                           |          |
|       |                                                                                                                                                                                 |                                           |          |
|       |                                                                                                                                                                                 |                                           |          |

Студент(ка) Антон БОРОВИК  
( підпис ) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи) Степан СКРУПСЬКИЙ  
( підпис ) (Ім'я ПРИЗВИЩЕ)

## РЕФЕРАТ

ПЗ: 113 с., 23 рис., 22 лістинги, 7 табл., 21 джерел.

### АВТОМАТИЗАЦІЯ, КОМП'ЮТЕРНА СИСТЕМА, МОДЕЛЮВАННЯ, ПРОЄКТУВАННЯ, ТУРИСТИЧНА ПОДОРОЖ

Об'єкт розробки — комп'ютерна система моделювання ефективного розподілу ваги раціону між учасниками туристичної подорожі.

Ціль даної магістерської роботи є дослідження та розробка спеціалізованої комп'ютерної системи для автоматизації процесу розподілу продуктів, враховуючи потреби користувачів і оптимізацію ресурсів.

Завдання магістерської роботи — створення клієнт-серверної системи для управління у туристичних походах, з акцентом на зручність і точність розрахунків.

Проект складається з 3 розділів:

Перший розділ містить аналіз існуючих рішень і технологій, актуальність автоматизації розподілу ваги, а також огляд архітектури клієнт-серверних систем та технологій для реалізації серверної та клієнтської частин.

Другий розділ описує розробку системи, визначення технічних вимог, вибір стеку технологій, структуру бази даних і основні функції.

Третій розділ описує реалізацію програми та результатам тестування, демонструє роботу системи, приклади коду та результати перевірки функціональності.

Результатом магістерської роботи є створення комп'ютерної системи яка забезпечує автоматизацію розподілу ваги продуктів між учасниками туристичних походів. Вона дозволяє зменшити помилки в плануванні і спрощує підготовку.

## ABSTRACT

PP: 113p., 23 figures, 22 listings, 7 table. 21 sources.

### MODELING, AUTOMATION, COMPUTER SYSTEM, TOURIST TRIP, DESIGN

The object of development is a computer system for modeling the effective weight distribution of the ration between participants of a tourist trip.

The goal of this master's thesis is to research and develop a specialized computer system for automating the process of product distribution, taking into account user needs and resource optimization.

The task of the master's thesis is to create a client-server system for managing tourist trips, with an emphasis on convenience and accuracy of calculations.

The project consists of 3 sections:

The first section contains an analysis of existing solutions and technologies, the relevance of weight distribution automation, as well as a review of the architecture of client-server systems and technologies for implementing the server and client parts.

The second section describes the development of the system, definition of technical requirements, selection of a technology stack, database structure and basic functions.

The third section describes the implementation of the program and the results of testing, demonstrates the operation of the system, code examples and the results of functionality testing.

The result of the master's thesis is the creation of a computer system that provides automation of the weight distribution of products between participants of tourist trips. It allows to reduce errors in planning and simplifies preparation.

## ЗМІСТ

|                                                                                                       |    |
|-------------------------------------------------------------------------------------------------------|----|
| Перелік умовних скорочень .....                                                                       | 9  |
| Вступ.....                                                                                            | 10 |
| 1 Аналіз існуючих рішень і технологій.....                                                            | 13 |
| 1.1 Терміни і визначення харчування в спортивному туризмі.....                                        | 13 |
| 1.2 Визначення завдання.....                                                                          | 15 |
| 1.3 Огляд існуючих рішень для розрахунків харчування в поході.....                                    | 16 |
| 1.3.1 Застосунок Calories in food.....                                                                | 17 |
| 1.3.2 The Ultimate Backpacking Calorie & Food Weight Calculator: Plan Your Next Outdoor Camping ..... | 17 |
| 1.3.3 Сайт ЯБ Поїла.....                                                                              | 17 |
| 1.4 Актуальність розробки системи ефективного розподілення ваги раціону ...                           | 18 |
| 1.5 Аналіз існуючих рішень побудови комп'ютерної системи .....                                        | 20 |
| 1.5.1 Основні архітектурні підходи .....                                                              | 20 |
| 1.6 Аналіз технологій для серверної частини системи.....                                              | 22 |
| 1.6.1 Основні технології серверної частини .....                                                      | 22 |
| 1.7 Аналіз технологій для клієнтської частини системи.....                                            | 24 |
| 1.7.1 Основні технології клієнтської частини .....                                                    | 24 |
| 1.7.2 Порівняння технологій .....                                                                     | 27 |
| 1.8 Висновки за розділом.....                                                                         | 27 |
| 2 Розробка Клієнт-серверної системи .....                                                             | 30 |
| 2.1 Етапи розробки системи .....                                                                      | 30 |
| 2.1.1 Розробка серверної частини .....                                                                | 30 |

|                                                                |    |
|----------------------------------------------------------------|----|
| 2.1.3 Розгортання та підтримка.....                            | 31 |
| 2.2 Визначення технічних вимог до системи .....                | 32 |
| 2.2.1 Вимоги розгортання і навантаження системи.....           | 33 |
| 2.2.2 Процесорні ресурси.....                                  | 33 |
| 2.2.3 Оперативна пам'ять .....                                 | 34 |
| 2.2.4 Сховище даних .....                                      | 34 |
| 2.2.5 Мережеві інтерфейси .....                                | 35 |
| 2.2.6 Операційна система .....                                 | 36 |
| 2.2.7 Вимоги до робочих характеристик клієнтської частини..... | 36 |
| 2.3 Визначення програмних вимог до системи .....               | 37 |
| 2.3.1 Визначення вимог до логічного устрою бази даних .....    | 38 |
| 2.3.2 Функціональність програмного забезпечення .....          | 39 |
| 2.3.3 Визначення вимог до інтерфейсу користувача системи ..... | 39 |
| 2.4 Функціональність інтерфейсу веб клієнта.....               | 40 |
| 2.5 Об'єкти системи .....                                      | 45 |
| 2.5.1 Сутності системи.....                                    | 45 |
| 2.5.2 Схема бази даних.....                                    | 49 |
| 2.6 Проектування архітектури системи .....                     | 50 |
| 2.6.1 Проектування репозиторія системи .....                   | 51 |
| 2.6.2 Проектування серверної частини .....                     | 52 |
| 2.6.3 Проектування клієнтського веб інтерфейсу.....            | 54 |
| 2.7 Розробка клієнтського веб інтерфейсу.....                  | 55 |
| 2.8 Розробка серверної частини .....                           | 56 |
| 2.9 Алгоритм пошуку ефективного розподілення .....             | 63 |

|                                                                                  |    |
|----------------------------------------------------------------------------------|----|
| 2.10 Висновки до розробки системи моделювання ефективного розподілення ваги..... | 71 |
| 3 Дослідження системи.....                                                       | 73 |
| 3.1 Підготовка системи до досліджень .....                                       | 74 |
| 3.2 Мета дослідження і організація експериментів .....                           | 77 |
| 3.3 Результати експериментів .....                                               | 78 |
| Висновки .....                                                                   | 85 |
| Перелік джерел посилання .....                                                   | 87 |
| Додаток А.....                                                                   | 89 |

## ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

SQL — Structured Query Language  
API — Application Programming Interface  
REST — Representational State Transfer  
GraphQL — Graph Query Language  
ORM — Object-Relational Mapping  
ACID — Atomicity, Consistency, Isolation, Durability  
MVCC — Multi-Version Concurrency Control  
DDD — Domain-Driven Design  
MVC — Model-View-Controller  
HTTP — HyperText Transfer Protocol  
UI — User Interface  
PWA — Progressive Web Applications  
NPM — Node Package Manager  
DOM — Document Object Model  
JSON — JavaScript Object Notation  
Docker — (назва платформи, не є скороченням)

## ВСТУП

Спортивний туризм швидко стає популярним серед людей різного віку. В епоху постійного стресу та швидкого ритму життя гірський та пішохідний туризм пропонує не лише фізичне навантаження, а й можливість усамітнення з природою, що допомагає відновити внутрішній баланс та покращити здоров'я. Для багатьох учасників подорож горами або лісовими стежками є унікальним способом перевірити свою витривалість, здатність працювати в команді та здобути навички виживання в природних умовах. Однак для досягнення успіху в подібних заходах необхідно особливо уважно підійти до планування, зокрема до аспектів харчування, яке є критично важливим у підтримці фізичної витривалості та здоров'я команди протягом усього походу.

Поширення спортивного туризму позитивно впливає на суспільство, сприяючи зміцненню фізичного та психічного здоров'я громадян, розвитку навичок командної роботи та формуванню екологічної свідомості. Зростання екологічних загроз спонукає до пошуку способів, що можуть покращити ставлення людей до природи, а подорожі та походи формують екологічну відповідальність і навчають раціонально використовувати природні ресурси. Проте для досягнення поставлених цілей спортивного туризму недостатньо просто вийти на маршрут. Успіх і безпека походу значною мірою залежать від того, наскільки правильно заплановано харчування, яке забезпечує учасників достатньою енергією та зберігає їхній фізичний і моральний стан [1].

Крім того, спортивний туризм сприяє соціалізації, адже об'єднує людей із різними інтересами, які хочуть розширити свої межі можливостей, перевірити витривалість і разом долати природні перешкоди. Гірські та пішохідні походи є унікальними способами пізнання себе, спілкування з однодумцями та отримання досвіду роботи в команді, що є цінним у різних сферах життя.

Правильне планування харчування — це один із ключових аспектів підготовки до спортивного походу, особливо коли мова йде про багатоденний

маршрут у горах або на природі. Раціон туристів повинен забезпечити їм усі необхідні поживні речовини та калорії, оскільки фізичне навантаження у природних умовах значно вище, ніж у повсякденному житті. Неправильне харчування або недостатнє споживання калорій може призвести до швидкої втоми, втрати сили та навіть зневоднення, що ускладнить похід та підвищить ризик непередбачуваних ситуацій.

Однією з основних вимог до харчування в поході є висока калорійність продуктів при їх невеликій вазі. Це дозволяє знизити навантаження на туристів, адже продукти необхідно нести на собі, часто у важких умовах пересування. Важливо продумати розподіл продуктів між учасниками, враховуючи їхню фізичну підготовку та особисті потреби. Крім того, харчування має бути збалансованим за основними поживними елементами — вуглеводами, білками та жирами, щоб забезпечити організм енергією і сприяти його відновленню після фізичних навантажень.

Правильне харчування є одним із найскладніших аспектів підготовки до багатоденного походу. Організаторам і учасникам необхідно враховувати індивідуальні потреби кожного в калоріях та поживних речовинах, що можуть суттєво змінюватись залежно від щоденного навантаження. У поході фізична активність часто коливається від помірних переходів до значних зусиль під час подолання природних перешкод, крутих підйомів чи складного рельєфу. Відповідно, у дні з підвищеним навантаженням добова норма калорій і певних мікроелементів (зокрема, білків та вуглеводів) повинна бути збільшена, аби запобігти виснаженню організму [2].

Також важливо правильно визначити, які продукти підходять для умов походу. Вони мають бути легкими, зручними для транспортування та бути достатньо калорійними, особливо зважаючи на потреби в енергії. У маршруті, де можливість відпочинку та приготування їжі обмежена, важливо передбачити декілька висококалорійних перекусів - таких як сухофрукти, горіхи, шоколад та туристичні сублімати, що швидко поповнюють енергію.

Окрім вищезгаданого, треба не забувати про гарячі прийоми їжі, які під час багатоденного походу є особливо важливими, оскільки сприяють не лише фізичному, а й психологічному комфорту учасників. Зважаючи на обмежений час та ресурси для приготування їжі, варто заздалегідь обрати страви, які можна легко приготувати в похідних умовах, і при цьому вони забезпечать необхідний запас енергії. В умовах низьких температур чи підвищеної вологості гарячі страви не лише допомагають відновити сили, але й зігрівають організм, що є особливо важливим для підтримки здоров'я під час походу [3].

Ретельне планування гарячих страв також включає розподіл палива та розрахунок часу, необхідного для їх приготування. Як правило, гарячі прийоми їжі організовують вранці та ввечері, що дозволяє підтримувати стабільний рівень енергії протягом доби. Протягом дня передбачають легкі та енергетично насичені перекуси, які не потребують додаткової підготовки.

Важливим моментом є розподіл продуктів між учасниками. Зазвичай продукти розподіляють рівномірно, щоб забезпечити ефективне навантаження на кожного учасника. Враховуючи рівень підготовки кожного, вагу продуктів намагаються збалансувати так, щоб не перевантажити менш підготовлених туристів. Такий підхід дозволяє підвищити загальну ефективність команди і знизити ризик виснаження в найвідповідальніші моменти подорожі. Правильне розподілення продуктів також полегшує організацію спільних прийомів їжі і допомагає зберегти психологічний комфорт у групі.

Таким чином, успіх і безпека багатоденних походів значною мірою залежать від правильного і гнучкого планування харчування, яке враховує добові норми калорій та поживних речовин відповідно до навантаження. Найкращий вибір продуктів та ефективний розподіл їх між учасниками сприяють збереженню енергії, підтримують фізичний комфорт і моральний дух команди, що дозволяє успішно подолати всі труднощі на маршруті.

# 1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ І ТЕХНОЛОГІЙ

## 1.1 Терміни і визначення харчування в спортивному туризмі

Спортивний туризм — це вид туристсько-спортивної діяльності, головною формою якої є спортивний похід або змагання з видів туризму, ці види діяльності спрямовані на випробовування фізичних, психічних, духовних сил людини, удосконалення туристської майстерності туриста, виконання нормативів спортивних розрядів. Спортивний туризм має за мету спортивне удосконалення в подоланні природних перешкод. Це означає удосконалення всього комплексу знань, умінь і навичок, фізичної підготовленості, необхідних для безпечного пересування людини по місцевості. Для реалізації спортивної мети здійснюють подорожі, насичені природними перешкодами, що дає змогу класифікувати їх як спортивні походи.

Спортивні туристичні походи — це групове або командне проходження туристичного спортивного маршруту певної протяжності з подоланням природних перешкод (перевалів, порогів, печер тощо) різної складності (рис 1.1) за нормативний час. В контексті цієї роботи важливо зазначити що похід має певну тривалість  $N$  днів, і проходиться командою з  $M$  учасників.

Харчова розкладка походу (в подальшому розкладка) — план харчування групи, або перелік готових страв, розподілених по прийомам їжі, з перерахуванням харчових продуктів, вказанням їх кількості, енергетичної цінності.

Розкладка складається з днів.

Кожен день містить кілька трапез. Наприклад сніданок, обід, вечеря, чи перекус.

Трапеза може включати складні страви, такі як борщ, чай з солодошами, або окремі продукти.

Вага розкладки — сумарна вага всіх продуктів разом з пакуванням, розрахованих на весь похід.

Вага розкладки на день - сумарна вага всіх продуктів разом з пакуванням, від вказаного дня до кінця походу. Наприклад вага розкладки на 5-й день 7-денного походу дорівнює вазі розкладки за 5, 6, та 7 день походу разом з пакуванням.

Пакунок – набір продуктів запакований для зручного транспортування і використання в поході. Прикладом може бути пакунок для приготування борщу, який включає в себе всі сублімовані, або сушені овочі, необхідні для приготування борщу на вечерю для всіх учасників походу.

Вага пакунку – брутто вага всіх складових пакунку разом з пакувальним матеріалом. Для вищенаведеного пакунку борщу, то може бути сума всіх сушених овочів, та харчова плівка, в яку їх загорнули, таким чаном вага пакунку біде мало відрізнятись від суми вагів продуктів пакунку. Але у випадку рибної консерви вага пакунку «тушенка яловича» скадатименться з нетто маси яловичої тушенки і бляшаної банки.

Важливо розрізняти нетто вагу, яка використовується в розкладці для розрахування колорійності, від брутто ваги пакунків, які будуть використовуватись для розподілення ваги раціону в цій роботі.



Рисунок 1.1 - Заброска групи А. Лебедева на перевалі Зотова, Памір 2014р

## 1.2 Визначення завдання

Підготовка, розрахунок і розподілення раціону між учасниками туристичної подорожі є складним процесом. Завданням дипломної роботи є дослідження та розробка системи автоматизації розподілення продуктів раціону між учасниками спортивного походу. Необхідно розробити алгоритм ефективного розподілу ваги продуктової розкладки походу між учасниками туристичної подорожі.

Ефективним вважається такий розподіл продуктів, який має наступні характеристики:

- приблизно рівну вагу продуктів у кожного учасника походу на початку походу;
- приблизно рівну вагу продуктів у кожного учасника походу протягом кожного дня;
- вага продуктів кожного учасника протягом всіх днів має бути монотонно спадаючою;
- фізичний об'єм продуктів має бути приблизно рівним (порівняйте 1 кг гречки і 1 кг сухарів);

Метою наукової роботи є розробка алгоритму, ефективного за критеріями, зазначеними вище, який дозволить вирішити задачу підбору ефективного розподілення раціону серед великої кількості можливих розподілень, і побудови комп'ютерної системи для використання цього алгоритму.

Для ілюстрації складності розподілення розкладки наведу спрощений приклад. В таблиці 1.1 наведене розподілення ваги продуктів для семиденного походу, в якому беруть участь 5 учасників, та для наглядності наведений графік на рисунку 1.2. Для розуміння, розкладка подібного похода включає десятки пакунків, а спортивні походи на три тижні і більшу кількість учасників включатимуть сотні пакунків, що робить ручне вирішення задачі ефективного розподілення вкрай складною.

Таблиця 1.1 - розподілення ваги продуктів для семиденного походу

|   | Тарас | Петро | Сергій | Юля | Аня | Вага розкладки |
|---|-------|-------|--------|-----|-----|----------------|
| 1 | 6.1   | 6.2   | 5.7    | 5.5 | 5.5 | 29             |
| 2 | 5.1   | 5.2   | 5.2    | 4.7 | 4.8 | 25             |
| 3 | 4.3   | 4.2   | 4.3    | 3.9 | 3.8 | 20.5           |
| 4 | 3.1   | 3.3   | 3.5    | 3   | 3.1 | 16             |
| 5 | 2.5   | 2.5   | 2.5    | 2.2 | 2.3 | 12             |
| 6 | 1.7   | 1.7   | 1.6    | 1.5 | 1.5 | 8              |
| 7 | 0.9   | 0.9   | 1      | 0.6 | 0.6 | 4              |

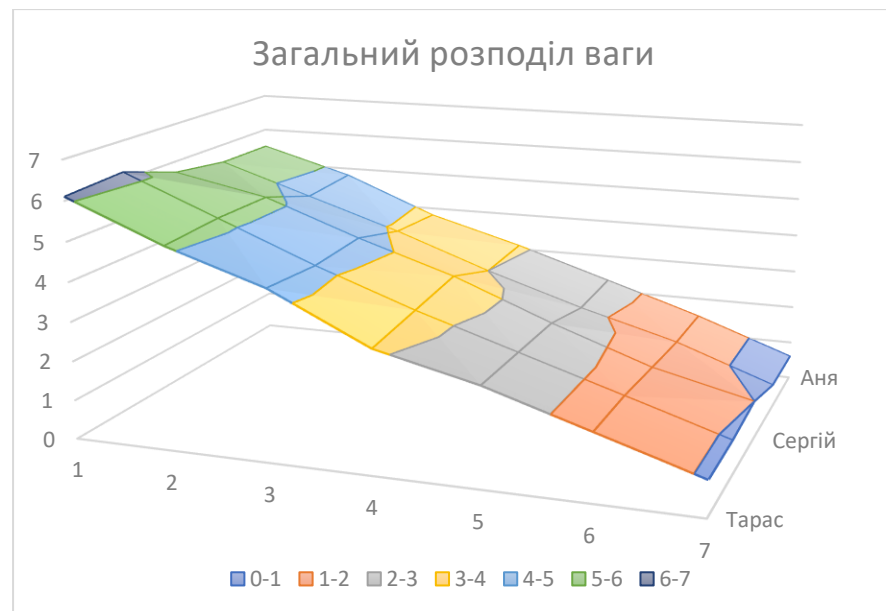


Рисунок 1.2 – графік розподілення ваги продуктів для семиденного походу

### 1.3 Огляд існуючих рішень для розрахунків харчування в поході

Сьогодні існує велика кількість веб-сайтів і мобільних застосунків, що спрямовані на вирішення завдань, пов'язаних з розрахунком калорійності харчування та підбором продуктів відповідно до заданих параметрів. Ці сервіси

пропонують користувачам широкий функціонал: від простих калькуляторів калорій до комплексних платформ, що враховують персональні параметри, такі як вік, вага, фізична активність та дієтичні потреби. Деякі з них також дозволяють налаштовувати індивідуальні харчові плани, зберігати історію споживання продуктів і відстежувати прогрес у досягненні цілей, пов'язаних зі здоров'ям або фітнесом.

### **1.3.1 Застосунок Calories in food**

Calories in food — один із численних застосунків, що надає доступ до таблиць калорійності продуктів. Завдяки цим даним користувачі можуть визначити енергетичну цінність найпоширеніших продуктів харчування та готових страв, що полегшує корекцію харчування під час туристичної подорожі відповідно до запланованих цілей. Проте цей застосунок не дозволяє організувати чи автоматизувати розподіл продуктів між учасниками походу, що є важливою складовою планування харчування для групи з урахуванням фізичних навантажень кожного учасника походу [4].

### **1.3.2 The Ultimate Backpacking Calorie & Food Weight Calculator: Plan Your Next Outdoor Camping**

Сайт Plan Your Next Outdoor Camping надає різноманітні калькулятори для точного планування харчування в походах: тут можна розрахувати загальну вагу всієї розкладки, середню вагу на одну особу та калорійність раціону. Крім цього, ресурс містить таблиці з прикладами розкладок для багатоденних походів, що допомагають орієнтуватися у виборі продуктів і їх кількості для подорожей різної тривалості. Однак, хоча сайт пропонує багато корисних інструментів для загального планування раціону, він не має можливості автоматизувати процес розподілу продуктів між учасниками, що є важливим аспектом для групових походів, де потрібно враховувати фізичні можливості кожного учасника [5].

### **1.3.3 Сайт ЯБ Поїла**

ЯБ Поїла — це зручний аналізатор продуктів, який дозволяє легко визначити кількість калорій, а також вміст білків, жирів і вуглеводів у продуктах. Калькулятор калорійності допомагає скласти збалансований раціон для

туристичного походу, а функція аналізатора рецептів дозволяє аналізувати інгредієнти та порівнювати поживну цінність готових страв. Крім того, сервіс пропонує зручну візуалізацію розкладки для походу. Проте, хоча він корисний для загального планування, ЯБ Поїла не надає можливості автоматично розподілити продукти між учасниками, що ускладнює управління раціоном у групових подорожах [6].

Попри таке різноманіття, більшість наявних рішень розраховані здебільшого на потреби індивідуального користувача та не враховують складні сценарії групового харчування або вимоги, що виникають під час багатоденних спортивних походів. Саме тому є потреба в створенні спеціалізованих рішень, що дозволять не лише розраховувати калорійність і поживні властивості продуктів, а й забезпечуватимуть інструменти для ефективного розподілення харчових ресурсів між учасниками в умовах підвищених фізичних навантажень і обмеженого доступу до джерел поповнення провіанту.

#### **1.4 Актуальність розробки системи ефективного розподілення ваги раціону**

Організація багатоденних походів потребує ретельного планування та врахування багатьох деталей, серед яких одним із найскладніших є розподіл ваги рюкзаків і, зокрема, продуктів харчування. Ефективний розподіл ваги раціону між учасниками походу не лише полегшує фізичне навантаження на кожного туриста, але й значно спрощує підготовку до подорожі. Саме тому питання автоматизації розподілу харчових продуктів настільки важливе, оскільки допомагає скоротити час, необхідний для ручних розрахунків, і знизити ймовірність помилок.

У багатоденних походах загальна вага продуктів, необхідних для забезпечення кожного учасника калоріями, значно зростає і її приблизні об'єми

ми можемо побачити на рисунку 1.3, тому ефективне планування цієї частини вантажу критично важливе. Учасники походу мають різну фізичну підготовку, і від ефективного розподілення ваги залежить рівномірне навантаження на кожного з них. Система, що автоматично визначає розподіл ваги харчових продуктів і враховує індивідуальні особливості кожного туриста (наприклад, фізичну підготовку, досвід, тип командного спорядження окрім продуктів у рюкзаках), дозволяє знизити ризик перевантаження і зробити похід більш комфортним.



Рисунок 1.3 - Набір продуктів перед пакуванням для гірського походу 4 категорії складності, Тянь-Шань 2012р.

Таким чином, розробка автоматизованої системи розподілу ваги продуктів харчування для туристичних походів є актуальною і корисною, оскільки забезпечує ефективну підготовку, рівномірне навантаження учасників і економію часу. Така система зменшує ручну роботу, робить похід більш комфортним і безпечним, а також дозволяє швидше адаптуватися до змін у маршруті або складі команди. Ефективний розподіл ваги раціону дає змогу ефективно використовувати енергію учасників і сприяє загальному успіху походу.

## **1.5 Аналіз існуючих рішень побудови комп'ютерної системи**

Розробку будь-якої системи необхідно починати з проектування. Саме на цьому кроці закладаються основи для подальшої реалізації, що допомагає уникнути проблем на наступних етапах роботи.

Створення складної комп'ютерної системи, є основою сучасних застосунків, які використовуються у багатьох сферах — від фінансових установ до вебсервісів і мобільних застосунків. Така система повинна забезпечувати надійну роботу, масштабованість та безпеку, що ставить перед архітекторами низку викликів. Розглянемо кілька основних архітектурних підходів, які допомагають досягти поставлених цілей при побудові таких систем. Ключові аспекти аналізу включають взаємодію між компонентами, розподіл навантаження та забезпечення безперервної доступності сервісу.

### **1.5.1 Основні архітектурні підходи**

#### **1.5.1.1 Клієнт-серверна архітектура (Client-Server Architecture)**

Класична клієнт-серверна архітектура передбачає наявність сервера, який відповідає за обробку запитів клієнтів і взаємодіє з базою даних для отримання або збереження даних. Клієнтська частина у цьому випадку відповідає за надання інтерфейсу користувачу та виконання первинної обробки даних. Серверна частина обробляє основну логіку, керує запитами до бази даних і забезпечує доступ до відповідних ресурсів.

Використання цього підходу дозволяє знизити навантаження на клієнтську частину, а також забезпечити централізований контроль доступу до даних та їх обробку. Недоліком такої архітектури є її обмежена масштабованість, оскільки для обробки великої кількості клієнтів потрібне додаткове горизонтальне або вертикальне масштабування серверів [7].

#### **1.5.1.2 Трирівнева архітектура (Three-Tier Architecture)**

Трирівнева архітектура передбачає розділення функціоналу системи на три рівні: клієнтський, серверний і базовий (рівень бази даних). На клієнтському

рівні здійснюється взаємодія з користувачем, серверний рівень відповідає за обробку бізнес-логіки, а рівень бази даних — за зберігання та управління даними. Такий підхід дозволяє легше управляти кожним з рівнів окремо і розділяти навантаження між ними, що полегшує масштабування і підвищує гнучкість системи.

Основною перевагою трирівневої архітектури є її здатність до масштабування та підтримка різних клієнтських платформ, таких як мобільні та десктопні клієнти, за допомогою використання єдиної серверної частини. Проте, якщо система потребує швидкої взаємодії з користувачем, то через збільшену кількість рівнів можуть виникнути затримки [8].

#### 1.5.1.3 Мікросервісна архітектура (Microservices Architecture)

Мікросервісна архітектура є розподіленою системою, у якій серверна частина розбивається на малі незалежні сервіси, кожен з яких виконує певний набір функцій і має чітко визначені інтерфейси взаємодії. Кожен сервіс працює окремо і може мати свою базу даних, що дозволяє обмежити вплив на інші сервіси у випадку відмови одного з компонентів.

Мікросервіси дають можливість масштабувати лише ті частини системи, що цього потребують, завдяки чому можна досягати значної гнучкості в розподілі ресурсів. Крім того, кожен сервіс може бути розроблений на своїй технологічній платформі. Водночас, мікросервісна архітектура ускладнює управління і потребує розробки додаткових рішень для моніторингу та координації, що є її недоліком [9].

#### 1.5.1.4 Архітектура «сервер-клієнт-база даних» з використанням REST або GraphQL

Цей підхід поєднує клієнт-серверну архітектуру з розширеними можливостями запитів до даних за допомогою REST або GraphQL. У такій архітектурі клієнт може запитувати дані в будь-якій потрібній йому структурі, а серверна частина використовує REST або GraphQL для передачі даних із бази. Це забезпечує високий рівень гнучкості та знижує кількість зайвих запитів.

Використання GraphQL зменшує навантаження на серверну частину, оскільки клієнт може отримувати всі потрібні йому дані одним запитом, без необхідності робити багаторазові звернення. Це ідеально підходить для застосунків з великою кількістю різних клієнтських платформ, що потребують різноманітних даних. REST API часто використовують для створення швидких і масштабованих вебзастосунків, які підтримують багатоплатформну роботу [10].

## **1.6 Аналіз технологій для серверної частини системи**

Сучасна серверна частина системи виконує важливу роль в архітектурі будь-якого програмного забезпечення. Вона забезпечує збереження, обробку та управління даними, а також підтримує бізнес-логіку, яка визначає функціональність застосунку. Серверні технології швидко розвиваються, і з урахуванням вимог до масштабованості, надійності та безпеки, вибір технологій для серверної частини є критично важливим для побудови продуктивних та стійких систем. Нижче розглянуто кілька провідних серверних технологій та їхні переваги для різних типів застосунків.

### **1.6.1 Основні технології серверної частини**

#### **1.6.1.1 Node.js**

Node.js — це середовище виконання JavaScript на сервері, яке дозволяє використовувати JavaScript для написання як серверної, так і клієнтської частини застосунків. Завдяки асинхронній, неблокуючій обробці запитів, Node.js добре підходить для застосунків з високим рівнем навантаження та реального часу, наприклад, чати, відеострімінг та ігрові сервери [11].

Основні переваги:

- висока продуктивність для обробки великої кількості одночасних з'єднань;
- підтримка асинхронних операцій;

- велика екосистема бібліотек NPM, що спрощує розробку.

#### 1.6.1.2 Django (Python)

Django — це фреймворк для серверної частини, написаний мовою Python. Він базується на принципі "Battery-included" та містить багато інструментів для швидкої розробки веб-додатків. Django часто використовується для побудови додатків з великою кількістю даних, наприклад, порталів з новинами чи соціальних мереж [12].

Основні переваги:

- вбудовані функції безпеки (захист від SQL-ін'єкцій, XSS тощо);
- потужна ORM (Object-Relational Mapping) для роботи з базами даних;
- велика спільнота і велика кількість доступної документації.

#### 1.6.1.3 Spring Boot (Java)

Spring Boot — це популярний Java-фреймворк для побудови серверних застосунків та мікросервісів. Він є частиною екосистеми Spring і забезпечує просту інтеграцію з іншими сервісами. Spring Boot підтримує як монолітні, так і мікросервісні архітектури та пропонує потужний набір інструментів для розробки корпоративних систем [13].

Основні переваги:

- висока масштабованість та стабільність;
- вбудовані механізми для роботи з REST API;
- підтримка мікросервісів через Spring Cloud.

#### 1.6.1.4 ASP.NET Core (C#)

ASP.NET Core — це фреймворк для серверної розробки від Microsoft, який працює на базі .NET Core і є кросплатформним. ASP.NET Core використовується для побудови веб-серверів, REST API та корпоративних систем. Завдяки ефективній інтеграції з Microsoft Azure, ASP.NET Core добре підходить для хмарних рішень [14].

Основні переваги:

- підтримка платформи Windows, Linux, macOS;

- висока продуктивність завдяки компіляції в машинний код;
- можливість інтеграції з іншими продуктами Microsoft.

#### 1.6.1.5 Ruby on Rails

Ruby on Rails — це фреймворк для розробки серверних додатків на Ruby, який активно використовують для побудови MVP (Minimum Viable Product) та прототипів завдяки швидкій розробці та гнучкості. Rails надає розробникам готові шаблони для побудови RESTful-додатків і працює за принципом "Convention over Configuration", що знижує складність налаштування [15].

Основні переваги:

- велика кількість вбудованих рішень для роботи з базами даних, безпекою та сесіями;
- швидкий розвиток продукту, зокрема для стартапів;
- гнучкість і масштабованість, що дозволяє працювати з різними типами застосунків.

### 1.7 Аналіз технологій для клієнтської частини системи

Розробка клієнтської частини сучасних програмних систем є важливою складовою успіху будь-якого проєкту. Клієнтська частина відповідає за взаємодію користувача із застосунком, надаючи доступ до функцій і даних у зручній та інтуїтивно зрозумілій формі. У 2020-х роках ключовими вимогами до клієнтських технологій стали швидкодія, гнучкість, адаптивність та можливість створення кросплатформних застосунків. Нижче ми розглянемо найбільш популярні технології для розробки клієнтської частини, та їхні.

#### 1.7.1 Основні технології клієнтської частини

##### 1.7.1.1 React

React — популярна бібліотека для створення інтерфейсів користувача, розроблена компанією Facebook. Вона дозволяє створювати швидкі,

компонентні інтерфейси з використанням декларативного підходу. React має великий екосистемний простір, що включає Next.js для серверного рендерингу, Redux для управління станом, і велика кількість інших бібліотек для спрощення розробки [16].

Основні переваги:

- компонентний підхід, який сприяє повторному використанню коду;
- висока продуктивність завдяки віртуальному DOM;
- широка підтримка та безліч доступних бібліотек.

#### 1.7.1.2 Angular

Angular — це фреймворк від Google для створення динамічних веб-застосунків, який включає в себе багато необхідних інструментів для повноцінного клієнтського рішення. Angular базується на TypeScript і пропонує потужні інструменти для управління станом, маршрутизації, валідації та багатьох інших задач [17].

Основні переваги:

- потужна архітектура та широкі можливості інтеграції;
- вбудована підтримка модульності, що спрощує розширення та масштабування;
- офіційна підтримка Google та активна спільнота.

#### 1.7.1.3 Vue.js

Vue.js — прогресивний фреймворк, який став популярним завдяки своїй простоті і зручності для швидкої розробки інтерфейсів. Vue.js поєднує в собі кращі концепції з React та Angular, пропонуючи гнучкий інструмент, який легко інтегрується з іншими бібліотеками або фреймворками [18].

Основні переваги:

- низький поріг входу для початківців;
- легка інтеграція в існуючі проекти;
- висока продуктивність та можливість створення адаптивних інтерфейсів.

#### 1.7.1.4 Svelte

Svelte — це інноваційний фреймворк, який відрізняється від інших підходом до рендерингу: він компілює компонент на етапі збірки, а не під час виконання. Це дозволяє створювати надзвичайно швидкі застосунки з мінімальним розміром коду [19].

Основні переваги:

- підвищена швидкість завантаження та виконання за рахунок компіляції;
- простіший синтаксис, що дозволяє створювати компоненти швидко і легко;
- відсутність необхідності віртуального DOM.

#### 1.7.1.5 Flutter (для кросплатформних застосунків)

Flutter — фреймворк від Google для створення кросплатформних застосунків із єдиною кодовою базою на мові Dart. Flutter дозволяє створювати застосунки з нативним виглядом і високою продуктивністю як для Android, так і для iOS, що робить його привабливим вибором для мобільних рішень [20].

Основні переваги:

- висока продуктивність завдяки компіляції в нативний код;
- велика кількість готових віджетів для побудови UI;
- кросплатформність, що скорочує час і витрати на розробку.

#### 1.7.1.6 Progressive Web Applications (PWA)

PWA — це підхід до розробки, який дозволяє веб-застосункам працювати подібно до нативних мобільних застосунків, включаючи офлайн-режим, пуш-сповіщення та доступ до апаратних ресурсів. PWA можна створювати на базі будь-якого з перерахованих фреймворків, таких як React, Angular чи Vue [21].

Основні переваги:

- можливість працювати офлайн та взаємодіяти з апаратними можливостями;
- простота розгортання та оновлення, оскільки це веб-технології;

- зменшення витрат на розробку мобільних додатків, оскільки створюється єдина кодова база для веб- і мобільного застосунку.

### **1.7.2 Порівняння технологій**

Кожна з розглянутих технологій має свої особливості та призначення, які роблять її придатною для різних сценаріїв розробки:

- висока продуктивність та кросплатформність: Flutter та Svelte ідеально підходять для високопродуктивних застосунків;
- модульність та підтримка великих проектів: Angular і React з Redux дозволяють масштабувати систему та організовувати великі проекти;
- швидкість розробки та адаптивність: Vue.js та PWA надають можливість швидкого створення простих та адаптивних застосунків;
- Інноваційність: Svelte пропонує новий підхід, що робить його актуальним для експериментальних рішень.

## **1.8 Висновки за розділом**

У першому розділі було детально проаналізовано значення ретельного планування харчування для успішного проведення спортивних походів. Було підкреслено, що підготовка раціону для багатоденних маршрутів є складним і важливим процесом, оскільки він повинен враховувати фізичне навантаження учасників, тип і тривалість переходів, добові норми калорійності, а також баланс необхідних поживних речовин. Окрему увагу було приділено складності ефективного розподілення ваги продуктів між учасниками з урахуванням етапів маршруту та наявності гарячих прийомів їжі, що потребує точних розрахунків.

Аналіз показав актуальність розробки комп'ютерної системи, яка здатна автоматизувати ці процеси, зменшуючи потребу в ручних розрахунках та прискорюючи підготовку до походу. Така система могла б забезпечити оптимізацію раціону, враховуючи змінні фактори, що впливають на витрати

енергії і доступність ресурсів, а також дозволила б рівномірно розподіляти вагу продуктів між усіма учасниками, значно підвищуючи ефективність організації спортивних походів.

Проаналізувавши існуючі архітектурні рішення для побудови складних комп'ютерних систем, було прийнято рішення застосувати трирівневу клієнт-серверну архітектуру. Враховуючи специфіку системи, зокрема відносно невелику кількість користувачів, які будуть одночасно активно її використовувати, обраний підхід забезпечує найліпший баланс між продуктивністю, масштабованістю та зручністю підтримки.

Трирівнева архітектура дозволяє чітко розмежувати клієнтський, серверний та базовий рівні, що спрощує оновлення та модифікацію кожного окремого компонента системи без впливу на інші. Клієнтська частина відповідатиме за інтерфейс і зручність використання для кінцевих користувачів, забезпечуючи можливість гнучкого управління раціоном і розподілом продуктів. Серверна частина оброблятиме бізнес-логіку, забезпечуючи зв'язок між клієнтом і базою даних та управління всіма операціями, пов'язаними з обробкою та розрахунком інформації. База даних забезпечуватиме надійне зберігання інформації про маршрути, харчові норми та необхідні ресурси для кожного учасника походу.

Для передачі даних між клієнтською та серверною частинами було обрано технологію REST, що є ефективним та масштабованим рішенням для асинхронного обміну інформацією. REST дозволяє легко реалізувати взаємодію між компонентами системи завдяки використанню стандартних HTTP-запитів, що знижує складність інтеграції і спрощує тестування. Такий підхід сприятиме стабільній та швидкій роботі системи, дозволяючи учасникам походу отримувати актуальну інформацію про розподіл ваги та раціон в режимі реального часу.

При виборі технологій для серверної частини комп'ютерної системи особлива увага приділялася забезпеченню високої продуктивності, підтримці гнучкого масштабування та зручності розгортання в хмарному середовищі.

Важливим аспектом також були надійні та ефективні вбудовані механізми для роботи з REST API, що забезпечують безперебійний обмін даними між сервером і клієнтом, необхідний для роботи системи в реальному часі.

Зважаючи на ці критерії, було обрано використання Spring Boot, який є частиною екосистеми Java. Spring Boot пропонує потужну підтримку для створення REST API завдяки своїм модульним компонентам та вбудованим інструментам, що оптимізують розробку бізнес-логіки і забезпечують обробку запитів з мінімальними затримками. Ця технологія також підтримує легке розгортання в різноманітних хмарних середовищах, таких як AWS, Google Cloud чи Microsoft Azure, що забезпечує гнучкість і зручність масштабування.

Крім того, Spring Boot має велику кількість бібліотек і модулів для безпечного управління даними, що дозволяє системі надійно зберігати та обробляти інформацію про маршрути, учасників та їхні харчові потреби. Завдяки цим перевагам Spring Boot став найбільш ефективним вибором для побудови серверної частини, здатної підтримувати необхідну функціональність та забезпечувати стабільну роботу системи в умовах змінного навантаження.

Під час вибору технологій для клієнтської частини комп'ютерної системи основними критеріями стали швидкість розробки, гнучкість та зручність використання для розробників різного рівня підготовки. Оскільки система має на меті надати користувачам інтуїтивно зрозумілий інтерфейс для роботи з розподілом раціону та плануванням походу, важливо було забезпечити технологію, яка дозволить швидко створювати адаптивний і динамічний інтерфейс.

Зважаючи на ці вимоги, було обрано Vue.js — прогресивний JavaScript-фреймворк, відомий своїми низькими вимогами та високою швидкістю розробки. Vue.js дозволяє створювати компонентний інтерфейс, що полегшує повторне використання елементів та підтримку коду, а також прискорює розробку за рахунок простого синтаксису та гнучкості у налаштуваннях. Інструменти Vue.js також полегшують інтеграцію з іншими бібліотеками та фреймворками, що дає змогу розширювати функціонал у майбутньому.

Vue.js забезпечує ефективну реактивність інтерфейсу, дозволяючи миттєво оновлювати дані, що є важливим для користувачів, які потребують оперативної інформації про розподіл ваги, харчування та змінні фактори, що впливають на планування маршруту. Завдяки цьому вибору, система отримує гнучкий, продуктивний і водночас простий у використанні клієнтський інтерфейс, який підтримуватиме ефективну роботу та легку адаптацію для нових користувачів.

## **2 РОЗРОБКА КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ**

### **2.1 Етапи розробки системи**

Розробка клієнт-серверної архітектури — це багаторівневий процес, що включає розробку серверної частини (логіка, безпека, API), клієнтського веб-інтерфейсу, контроль якості, розгортання та підтримку. Кожен етап потребує ретельного і детального планування. Одним з найважливіших етапів є проектування архітектури системи, який включає в себе наступні компоненти:

- серверна частина: визначення серверного середовища (хмарні рішення, локальні сервери тощо), бази даних, API;
- клієнтська частина: вибір технологій для фронтенда (HTML/CSS, JavaScript, фреймворки на зразок React, Vue або Angular);
- планування взаємодії між клієнтом і сервером (REST, GraphQL, WebSocket).

#### **2.1.1 Розробка серверної частини**

Сервер — основа системи, яка обробляє запити, забезпечує безпеку і зберігає дані. Основні етапи розробки сервера:

- а) Вибір стеку технологій:
  - 1) мови програмування: Python (Django, Flask), Node.js, Java (Spring), Ruby, PHP;
  - 2) СУБД: PostgreSQL, MySQL, MongoDB.

- б) Розробка API:
  - 1) проєктування ендпоінтів;
  - 2) реалізація логіки обробки запитів (CRUD-операції, авторизація, обробка помилок).
- в) Інтеграція бази даних:
  - 1) створення моделей і таблиць;
  - 2) налаштування ORM (Object-Relational Mapping).
- г) Модульне тестування:
  - 1) Тестування окремих компонентів серверної частини (Unit tests).

### **2.1.2 Розробка клієнтської частини (веб-інтерфейс)**

Клієнтська частина забезпечує зручний інтерфейс для користувачів.

Основні етапи розробки клієнтської частини:

- а) Прототипування:
  - 1) створення макетів (Figma, Adobe XD);
  - 2) UX/UI-дизайн для забезпечення зручності користування.
- б) Розробка:
  - 1) використання фреймворків: React, Angular, Vue.js;
  - 2) адаптивна верстка (HTML5, CSS3, SCSS);
  - 3) інтеграція з сервером через API.
- в) Оптимізація продуктивності:
  - 1) оптимізація розміру сторінок;
  - 2) Lazy loading ресурсів.
- г) Тестування клієнтської частини:
  - 1) перевірка на сумісність із браузерами;
  - 2) коректність взаємодії з сервером.

### **2.1.3 Розгортання та підтримка**

Розгортання полягає в налаштуванні середовища виконання для сервера та клієнта і містить наступні кроки:

- Вибір середовища - локальні сервери, хмарні сервіси (AWS, Google Cloud, Azure, Heroku).
- Контейнеризація - використання Docker/Kubernetes для спрощення розгортання.
- CI/CD - налаштування автоматичного розгортання через Jenkins, GitHub Actions або GitLab CI.
- Налаштування домену та SSL - забезпечення безпечного доступу.

## **2.2 Визначення технічних вимог до системи**

Визначення технічних вимог до клієнт-серверної системи є важливим етапом, що забезпечує стабільність, продуктивність і масштабованість проєкту. Треба приділити увагу вимогам до процесора, оперативної пам'яті, сховища даних, мережевих інтерфейсів, та операційної системи.

Потужність процесора визначає, як ефективно сервер оброблятиме запити, виконуватиме бізнес-логіку та підтримуватиме взаємодію з клієнтами. Оперативна пам'ять впливає на швидкість обробки даних і стабільність системи, особливо при одночасній роботі з великою кількістю користувачів. Вибір сховища даних (SSD або HDD) та його обсягу залежить від потреб системи, включаючи бази даних, резервне копіювання та журналювання. Мережеві інтерфейси відіграють ключову роль у забезпеченні швидкості передачі даних між клієнтом і сервером. Сервер має підтримувати високу пропускну здатність мережі, а також резервні канали для безперервної роботи.

Важливим є вибір операційної системи: сервери на Linux/UNIX пропонують високу надійність і гнучкість, тоді як Windows Server підходить для специфічних завдань. На стороні клієнта необхідно враховувати сумісність з основними ОС та оптимізацію роботи для пристроїв із низькою продуктивністю.

Для економічної ефективності проекту ключовим фактором є масштабованість інфраструктури та прогнозування майбутніх навантажень. Система повинна легко адаптуватися до збільшення кількості користувачів і обсягу даних. Правильне визначення технічних вимог мінімізує ризики, дозволяє уникнути непередбачуваних витрат і забезпечує довгострокову ефективність клієнт-серверної архітектури.

### **2.2.1 Вимоги розгортання і навантаження системи**

Система розроблена з урахуванням специфічних потреб вузької категорії користувачів — керівників багатоденних туристичних походів. Вона орієнтована на надання інструментів для планування та обліку ресурсів і управління даними, що не вимагає одночасного доступу великої кількості користувачів. Оскільки основними користувачами є окремі фахівці, а не великі групи або організації, система не потребує масштабного хмарного розгортання, яке зазвичай призначене для роботи з багатьма користувачами в режимі реального часу.

Натомість цілком достатньо розгорнути систему на локальному сервері або навіть на особистому ноутбучі користувача. Такий підхід забезпечить зручність і автономність роботи, що є важливим у контексті туристичних походів, де доступ до Інтернету може бути обмеженим. Однокористувацький режим роботи дозволить керівнику походу працювати з системою без необхідності постійного підключення до мережі, що сприяє її адаптованості до умов польового використання.

### **2.2.2 Процесорні ресурси**

Для типового розгортання серверної частини системи, на особистому ноутбучі, основними вимогами є ефективне співвідношення продуктивності та ресурсоемності. У таких випадках критично важливими є багатоядерний процесор і достатній запас потужності для одночасного виконання серверної логіки та бази даних без значних затримок.

Рекомендованими є процесори із чотирма ядрами і тактовою частотою 2.5–3.0 ГГц, такими як Intel Core i5 або AMD Ryzen 5. Для забезпечення стабільної роботи системи важливо використовувати CPU з підтримкою багатопотоковості

(8+ потоків) і кешем L3 обсягом від 6 до 8 МБ. Такі параметри дозволяють ефективно обробляти складні SQL-запити і виконувати бізнес-логіку сервісу. Висока тактова частота сприяє плавній роботі навіть у разі пікових навантажень.

Мінімальні вимоги включають двоядерний процесор із тактовою частотою 2.0 ГГц і кешем L3 обсягом 3–4 МБ, наприклад, Intel Core i3 або AMD Ryzen 3. Цього достатньо, однак за таких умов можуть виникати затримки під час виконання великих обчислень або складних запитів до бази даних, тому для більш стабільної роботи рекомендовано мати додатковий запас потужності.

### **2.2.3 Оперативна пам'ять**

Для клієнт-серверних систем оперативна пам'ять є одним із ключових параметрів, що визначає продуктивність і стабільність роботи. Сервер споживає значні ресурси, особливо під час обробки великих обсягів даних чи складних запитів. Типове локальне розгортання передбачає запуск усіх сервісів на одному пристрої, тому необхідно забезпечити достатній обсяг оперативної пам'яті для уникнення перевантажень.

Рекомендований обсяг оперативної пам'яті для комфортної роботи системи з середнім навантаженням становить 16 ГБ. Це дозволяє одночасно працювати з серверною логікою, базою даних і іншими програмами, що можуть бути активними на пристрої. Наприклад, PostgreSQL потребує приблизно 4–6 ГБ для ефективного кешування та обробки запитів, а Spring Boot сервіс із середньою складністю логіки може займати від 2 до 4 ГБ. Решта обсягу RAM необхідна для операційної системи, браузера та інших програм, що працюватимуть паралельно.

Мінімально необхідний обсяг оперативної пам'яті — 8 ГБ. У таких випадках система буде працювати стабільно, однак можуть виникати затримки під час пікових навантажень або обробки великих обсягів даних. Для тестових середовищ і розробки цього може бути достатньо, але для продуктивного використання рекомендовано мати щонайменше 16 ГБ RAM.

### **2.2.4 Сховище даних**

Клієнт-серверна система, потребує продуктивного та надійного сховища даних для ефективної роботи. Вибір типу сховища та його параметрів впливає на

швидкість виконання запитів до бази даних, час запуску серверного додатка та загальну стабільність системи. Для локального розгортання на ноутбуці або сервері важливо враховувати обсяг і швидкодію сховища, щоб уникнути затримок і втрат продуктивності.

Рекомендованим є використання SSD-накопичувачів (Solid State Drive), які забезпечують високу швидкість читання і запису даних. Для систем із середнім навантаженням достатньо SSD обсягом 256–512 ГБ, де близько 50–100 ГБ буде виділено під базу даних, решта — сервіса, резервних копій і операційної системи. Висока швидкодія SSD сприяє зменшенню часу виконання складних SQL-запитів, а також забезпечує швидке завантаження сервісу і операційної системи.

Мінімальні вимоги до сховища включають HDD (Hard Disk Drive) обсягом від 500 ГБ для зберігання бази даних і допоміжних файлів. Проте використання HDD може призводити до затримок під час виконання операцій із великими обсягами даних, що може впливати на продуктивність системи в цілому. У тестових середовищах HDD може бути прийнятним, але для продуктивної роботи та мінімізації часу відгуку системи бажано використовувати SSD.

### **2.2.5 Мережеві інтерфейси**

Для локального розгортання на ноутбуці, особливо в однокористувацькому режимі, вимоги до мережевих інтерфейсів є мінімальними. У таких випадках достатньо внутрішніх локальних з'єднань між сервісом і СКБД через localhost. Проте, якщо передбачається використання мережевого доступу, наприклад, для підключення веб клієнтів в локальній мережі, бажано мати Ethernet-порт або стабільний модуль Wi-Fi із підтримкою швидкості передачі даних не менше 100 Мбіт/с.

Для середовищ із більшим навантаженням, де система обслуговує декілька користувачів одночасно, рекомендовано використовувати Ethernet-з'єднання зі швидкістю 1 Гбіт/с для мінімізації затримок і забезпечення стабільної роботи. Мережеві інтерфейси повинні підтримувати резервування (наприклад, через використання декількох мережевих карт) для забезпечення безперервної роботи

у разі збоїв. Додатково слід налаштувати параметри брандмауера для захисту доступу до СКБД і сервісу.

### **2.2.6 Операційна система**

Операційна система для роботи системи, що включає сервіс і СКБД, має забезпечувати стабільність, сумісність і ефективне використання апаратних ресурсів. Вибір ОС залежить від цілей розгортання: локальна робота на ноутбучі, серверне середовище або хмарна інфраструктура. Для локального розгортання на ноутбучі або персональному комп'ютері важливо враховувати зручність налаштування й підтримку потрібного програмного забезпечення.

Рекомендованими є дистрибутиви Linux, такі як Ubuntu, Debian або CentOS, завдяки їх високій продуктивності, стабільності та широкій підтримці для серверних додатків. Більшість СКБД мають чудову сумісність із Linux-середовищем, що дозволяє зменшити затримки, оптимізувати роботу мережеских інтерфейсів і забезпечити надійність. Для локального використання на ноутбучі можна вибрати Linux-дистрибутив із графічним інтерфейсом для більшої зручності.

Для користувачів, які віддають перевагу Windows, можливе розгортання на Windows 10/11 (Pro або Enterprise) чи Windows Server. У такому випадку важливо переконатися, що обидва компоненти (сервіс та СКБД) правильно налаштовані та мають необхідні дозволи. Windows є менш оптимізованим для серверних завдань, ніж Linux, однак підходить для локальної розробки та тестування. Незалежно від обраної ОС, система повинна мати доступ до оновлень безпеки, стабільної роботи мережеских служб і налаштувань віртуальної пам'яті для ефективного управління ресурсами.

### **2.2.7 Вимоги до робочих характеристик клієнтської частини**

Клієнтський веб-інтерфейс системи розроблений таким чином, щоб бути доступним як із браузерів настільних пристроїв, так і з мобільних платформ (Android, iOS, HarmonyOS). Це забезпечує універсальність і зручність використання для різних категорій користувачів, незалежно від типу їхніх

пристроїв. Важливим аспектом є підтримка адаптивного дизайну та коректна робота інтерфейсу на екранах різних розмірів.

Браузер, що використовується для відкриття інтерфейсу, повинен підтримувати сучасні веб-технології та забезпечувати стабільну роботу. Оскільки клієнтська частина реалізується з використанням HTML5, CSS3 і JavaScript, рекомендується використовувати сучасні браузери, наприклад, Google Chrome, Mozilla Firefox, Microsoft Edge або Safari. Важливо, щоб версія браузера була актуальною, що гарантує підтримку нових функцій, високу продуктивність і оновлення безпеки.

Для роботи клієнтського інтерфейсу браузер повинен мати увімкнену підтримку JavaScript і WebSocket (якщо використовується функціонал реального часу). На мобільних пристроях бажано використовувати браузери, які регулярно оновлюються та підтримують усі основні веб-стандарти (наприклад, Chrome для Android або Safari для iOS). Уникати слід застарілих браузерів, таких як Internet Explorer, оскільки вони можуть спричиняти помилки в роботі інтерфейсу або не підтримувати необхідні функції.

## **2.3 Визначення програмних вимог до системи**

Визначення програмних вимог до системи є критично важливим етапом у розробці, оскільки саме вони задають основу для її функціонування та взаємодії з користувачами. Логічний устрій бази даних забезпечує структуру для зберігання та обробки даних, відображаючи зв'язки між сутностями та оптимізуючи їх доступ. Наприклад, правильно спроектована схема бази даних допомагає уникнути дублювання даних, забезпечити їхню цілісність і підвищити ефективність запитів. Недостатньо опрацьований устрій БД може спричинити проблеми продуктивності, ускладнення оновлень та труднощі з масштабуванням системи.

Функціональність і інтерфейс користувача визначають, наскільки зручно та ефективно система задовольняє потреби користувачів. Чітко сформульовані вимоги до функціональних можливостей (наприклад, авторизація, фільтрація даних, звітність) гарантують, що кожна частина системи відповідає реальним задачам. Інтерфейс користувача повинен бути інтуїтивно зрозумілим і адаптованим до різних пристроїв, щоб забезпечити комфортну роботу як для новачків, так і для досвідчених користувачів. Відсутність чіткого визначення цих вимог може призвести до розробки, яка не відповідає очікуванням користувачів, або до функціоналу, що не використовується.

### **2.3.1 Визначення вимог до логічного устрою бази даних**

Визначення вимог до логічного устрою бази даних (БД) є фундаментальним етапом, що забезпечує правильну організацію зберігання та обробки даних. Один із ключових аспектів цього процесу — нормалізація даних, зокрема приведення їх до третьої нормальної форми (3НФ). Це дозволяє мінімізувати дублювання інформації, усунути аномалії вставки, видалення та оновлення, а також забезпечити цілісність і узгодженість даних. Наприклад, розділення таблиць із залежними атрибутами на окремі сутності дозволяє зберігати лише релевантну інформацію в кожній таблиці, що спрощує її підтримку та редагування.

При проектуванні БД важливо враховувати оптимізацію структури для ефективного процесу читання вкладених структур, таких як пов'язані таблиці. Використання індексів, зовнішніх ключів і чітко визначених зв'язків між сутностями дозволяє швидко та чітко виконувати складні запити, наприклад, отримання даних із кількох рівнів вкладеності таблиці. Для великих обсягів даних або багаторазових операцій читання можна також застосовувати денормалізовані структури (часткове порушення нормалізації) у вузьких місцях для досягнення найефективнішого балансу між швидкістю та гнучкістю.

Особливу увагу слід приділити чіткості та передбачуваності операцій додавання та редагування даних. Це включає валідацію введених значень, забезпечення цілісності зв'язків через каскадні оновлення або видалення, а також

уникнення дублювань за допомогою унікальних ключів. Коректно спроектована структура спрощує розширення системи та інтеграцію з іншими компонентами, водночас знизити ризики помилок і конфліктів даних у процесі роботи.

### **2.3.2 Функціональність програмного забезпечення**

Система моделювання ефективного розподілення ваги раціону між учасниками туристичної подорожі має виконувати наступні функції:

- відображення довідника продуктів (довідник включає продукти, страви, типи трапез, категорії продуктів, категорії страв);
- редагування довідника продуктів (додавання нових елементів, зміна існуючих і видалення непотрібних);
- відображення розкладки туристичної подорожі;
- додавання, видалення туристичної розкладки;
- відображення учасників туристичної подорожі;
- редагування учасників туристичної подорожі;
- відображення пакунків обраної розкладки;
- редагування пакунків обраної розкладки;
- редагування днів обраної розкладки (включає в себе редагування трапез, страв і продуктів);
- редагування трапез обраного дня розкладки (включає в себе редагування страв і продуктів);
- призначення продукту до обраного пакунку розкладки (необхідно для роботи алгоритму розподілення);
- експорт розкладки в Excel разом з побудовою ефективного розподілення продуктів між учасниками.

### **2.3.3 Визначення вимог до інтерфейсу користувача системи**

Проектування інтерфейсу користувача є важливою складовою розробки веб-клієнта. Він має бути інтуїтивно зрозумілим і забезпечувати простоту використання. Для цього необхідно враховувати потреби різних категорій користувачів та уникати перевантаження зайвими елементами. Наприклад, головна сторінка повинна мати чітку навігацію, а всі основні функції системи —

бути доступними у декілька кліків. Важливо приділити увагу зручному розташуванню кнопок, полів введення та інших елементів управління для полегшення взаємодії з системою.

Адаптивність UI є ключовою вимогою, оскільки сучасні веб-додатки використовуються на пристроях із різними розмірами екранів, від смартфонів до стаціонарних комп'ютерів. Інтерфейс повинен автоматично підлаштовуватися під розмір екрана, забезпечуючи збереження функціональності й читабельності. Використання адаптивного дизайну із застосуванням технологій CSS (Flexbox, Grid) та бібліотек типу Bootstrap дозволяє уникнути проблем із відображенням на мобільних пристроях чи планшетах.

Дизайн UI має бути не тільки зручним, але й візуально привабливим. Система повинна використовувати зрозумілу кольорову палітру, що не перевантажує зір користувача, а також забезпечувати достатній контраст для тексту й іконок. Ключові елементи (наприклад, кнопки дій, повідомлення про помилки або успішне виконання операцій) мають виділятися і бути помітними, щоб допомагати користувачам у швидкій взаємодії із системою.

Швидкість відгуку інтерфейсу також є важливим аспектом, адже користувачі очікують миттєвого реагування на їхні дії. Інтерфейс повинен передавати інформацію про обробку запитів (спінери, статус-бар або інші індикатори), щоб уникнути непорозумінь. Крім того, необхідно мінімізувати кількість непотрібних перезавантажень сторінок, використовуючи технології, такі як AJAX, WebSocket або інші засоби для динамічного оновлення вмісту.

## **2.4 Функціональність інтерфейсу веб клієнта**

На підставі вищенаведеного опису функціональності системи можна визначити наступну функціональність інтерфейсу веб клієнта:

ФІВК-1. Відображення довідника продуктів. Для переходу до довідників, користувач має обрати меню «Довідники» верхнього рівня. Далі відображається меню другого рівня з назвами елементів довідника. Обрання елемента меню другого рівня виводить на екран таблицю з елементами обраного довідника, наприклад за замовченням це продукти.

ФІКВ-2. Редагування довідника. Для редагування довідника треба обрати довідник, згідно ФІКВ-1. Для зміни або видалення елемента довіднику слід скористатись відповідними кнопками справа від елемента довідника в таблиці. Для додавання нового елемента треба скористатись кнопкою-посиланням «Додати новий» внизу таблиці елементів довідника.

Для прикладу, на рисунку 2.1 наведена схема послідовності створення та редагування страви:

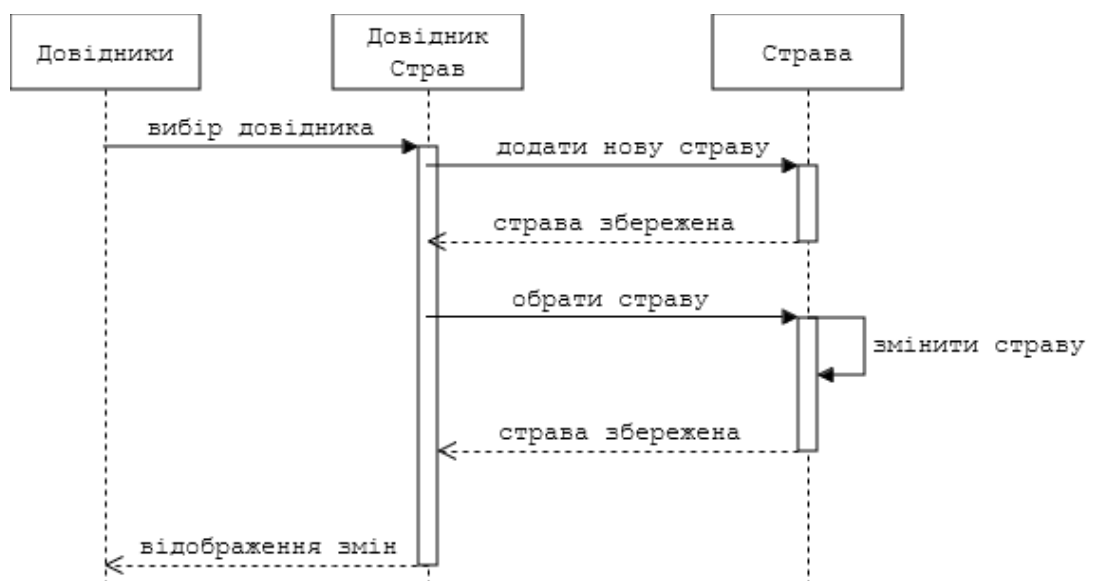


Рисунок 2.1 – Схема створення та редагування страв.

ФІКВ-3. Відображення розкладки туристичної подорожі. На домашній сторінці відображається список наявних розкладок. Назва розкладки є посиланням, яке веде до сторінки з повною інформацією про розкладку і можливістю редагувати розкладку. Ця сторінка містить: секцію з основною інформацією про розкладку, секцію учасників подорожі, секцію днів розкладки.

ФІКВ-4. Додавання, видалення туристичної розкладки. На домашній сторінці зі списком розкладок, для кожної розкладки є кнопка видалення. Видалити можна тільки пусту розкладку. Додати нову розкладку можна за допомогою кнопки-посилання «Додати нову розкладку» знизу списку розкладок.

ФІКВ-5. Відображення учасників туристичної подорожі. На сторінці з повною інформацією про розкладку (дививсь ФІКВ-3), під секцією з основною інформацією є секція учасників туристичної подорожі.

ФІКВ-6. Редагування учасників туристичної подорожі. На сторінці з повною інформацією про розкладку (дививсь ФІКВ-3), в секції учасників, для кожного учасника є кнопка «Змін.» для зміни інформації про учасника, згідно даних в таблиці і кнопка «Видал.» для видалення учасника. Додати нового учасника можна за допомогою кнопки-посилання «Додати учасника» знизу списку учасників.

ФІКВ-7. Відображення пакунків обраної розкладки. На сторінці з повною інформацією про розкладку (дививсь ФІКВ-3), правіше секції з основною інформацією є кнопка «Пакунки», яка веде на сторінку Пакунків обраної розкладки.

ФІКВ-8. Редагування пакунків обраної розкладки. На сторінці пакунків обраної розкладки (дививсь ФІКВ-7) для кожного пакунка є кнопка «Змін.» для зміни інформації про пакунок, згідно даних в таблиці і кнопка «Видал.» для видалення пакунку. Додати новий пакунок можна за допомогою кнопки-посилання «Додати пакунок» знизу списку пакунків.

ФІКВ-9. Редагування днів обраної розкладки (включає в себе редагування трапез, страв і продуктів). На сторінці з повною інформацією про розкладку (дививсь ФІКВ-3) в секції днів, кожен день містить кнопки «Видал.» для видалення дня і «Змін.», яка веде на сторінку редагування дня. Видалити можна лише порожній день. На сторінці редагування дня є секція з основною інформацією і кнопкою «Змінити», яка оновлює основну інформацію про день. Нижче розташовані: секція трапез, секція страв і секція продуктів обраного дня. Страви зі списку можна змінити і видалити за допомогою відповідних кнопок

праворуч. Продукти можна видалити і змінити їх порядок за допомогою кнопок праворуч. Редагування продуктів дивись ФІКВ-11. Ніжче розташовані кнопки посилання:

- «Показати страви», яка відображує секцію з довідником страв для вибору і додавання до дня;
- «Показати продукти», яка відображує секцію з довідником продуктів для вибору і додавання;
- «Додати трапезу», яка відображує секцію для додавання нової трапези.

ФІКВ-10. Редагування трапез обраного дня розкладки (включає в себе редагування страв і продуктів). На сторінці редагування дня (дививсь ФІКВ-9) в секції трапез, кожна трапеза містить кнопки «Видал.» для видалення трапези і «Змін.» яка веде на сторінку редагування трапези. На сторінці редагування трапези є секція з основною інформацією і кнопкою «Змінити», яка оновлює основну інформацію про трапезу. Нижче розташовані секція страв і секція продуктів обраної трапези. Страви зі списку можна змінити і видалити за допомогою відповідних кнопок праворуч. Продукти можна видалити і змінити їх порядок за допомогою кнопок праворуч. Редагування продуктів дивись ФІКВ-11. Ніжче розташовані кнопки посилання:

- «Показати страви», яка відображує секцію з довідником страв для вибору і додавання до дня;
- «Показати продукти», яка відображує секцію з довідником продуктів для вибору і додавання;

На рисунку 2.2 є приклад редагування страви визначеної трапези, а потім видалення трапези.

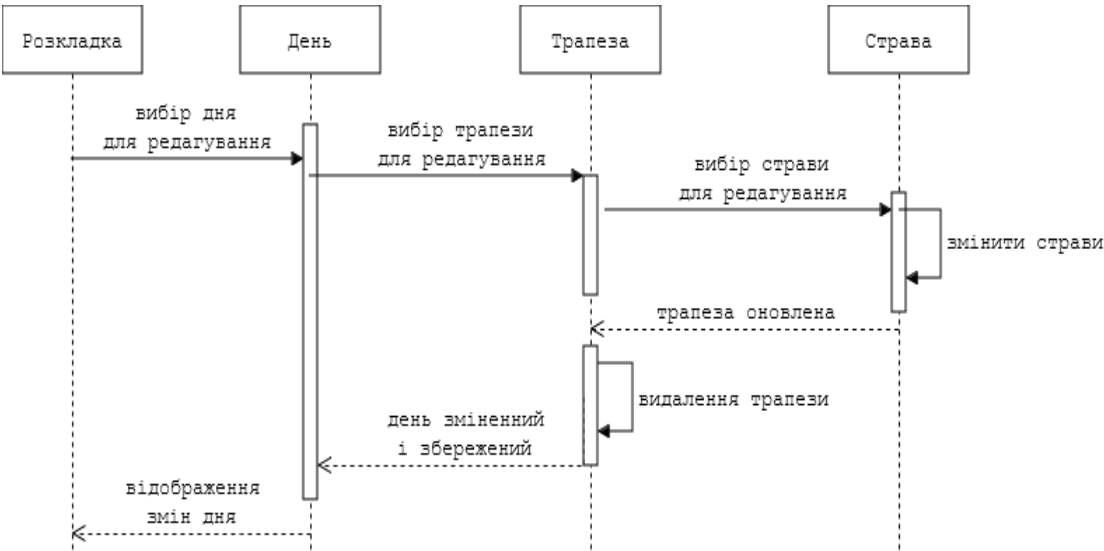


Рисунок 2.2 – редагування/видалення трапези

ФІКВ-11. Призначення продукту до обраного пакунку розкладки (необхідно для роботи алгоритму розподілення). На сторінках редагування дня, трапези і страви (ФІКВ-9,10) в секції продуктів кожен продукт має колонку «Пакунок». Ця колонка містить випадаючий список з назвами пакетів, створених для цієї розкладки. Для додавання продукту в пакунок треба обрати необхідний пакунок із списку і натиснути кнопку «Змінити» під основною секцією (рис 2.3).

|                        |       |      |      |      |    |          |
|------------------------|-------|------|------|------|----|----------|
| Цукор                  | 16    | 0    | 0    | 3.99 | 20 | цукор1   |
| Цукерки                | 3.5   | 0.03 | 0.07 | 0.85 | 10 | цукерки1 |
| <i>Рибний суп</i>      |       |      |      |      |    |          |
| рис                    | 16.75 | 0.34 | 0.05 | 3.64 | 25 | супи2    |
| картопля сушена        | 7.05  | 0.14 | 0    | 1.63 | 15 | супи2    |
| риба сушена            | 5.1   | 0.96 | 0.13 | 0    | 15 | супи2    |
| Цибуля ріпчаста сушена | 0.24  | 0.01 | 0    | 0.04 | 3  | супи2    |
| Морква сушена          | 0.1   | 0.01 | 0    | 0.02 | 2  | супи2    |
| Сіль                   | 0     | 0    | 0    | 0    | 3  | сіль     |
| Перець чорний          | 0     | 0    | 0    | 0    | 1  | спеції   |
| Зелень сушена          | 0     | 0    | 0    | 0    | 1  | спеції   |

Змін. Видал.

Продукти

| Назва                | Калорії | Протеїни | Жири | Вуглвд. | Вага | Пакунок |
|----------------------|---------|----------|------|---------|------|---------|
| сухарі білі пшеничні | 30      | 0.95     | 0.11 | 6.17    | 30   | сухарі2 |
| Загалом              | 79.64   | 2.43     | 0.35 | 16.54   | 140  |         |

Показати страви Показати продукти

^ v Видал.

Рисунок 2.3 – таблиця зміни продуктів

ФІКВ-12. Експорт розкладки в Excel разом з побудовою ефективного розподілення продуктів між учасниками. На сторінці з повною інформацією про

розкладку (дививсь ФІКВ-3), правіше секції з основною інформацією є кнопка «Експорт», яка експортує в ексель файл наступні сторінки:

- розкладку по днях;
- загальний список продуктів з вагою на 1 людину і на групу;
- список пакунків з продуктами;
- ефективне розподілення продуктів (список пакунків для кожного учасника на окремому аркуші).

## 2.5 Об'єкти системи

Відповідно до опису функцій інтерфейсу, у системі можна визначити такі основні сутності: Продукт, Посилання продукту, Страва, Категорії продукту і страви, Трапеза і Тип Трапези, День, Розкладка, Турист, Пакунок.

### 2.5.1 Сутності системи

Категорія Продукту – сутність належить до довіднику наведений у лістингу 2.1.

#### Лістинг 2.1 - ProductCategory

```
@Data
public class ProductCategory {
    private final long categoryId;
    private String name;
}
```

Продукт – одна з основних сутностей системи наведений у лістингу 2.2.

#### Лістинг 2.2 - Product

```
@Data
public class Product {
```

```

        private final long productId;
        private String name;
        private ProductCategory category;
        private float calorific; //калорії в кКалорії на 100 грам
        private float proteins; //протеїни в грамах на 100 грам
        private float fats; //жири в грамах на 100 грам
        private float carbs; //вуглеводи в грамах на 100 грам
        private int defaultWeight; //вага продукту за замовченням в
0.1 грама
    }

```

Посилання Продукту – об’єкт обгортка для зв’язування продукту зі стравою, трапезою або днем. Цей об’єкт дозволяє не зберігати однакові параметри продукту для кожного посилання наведені у лістингу 2.3

### Лістинг 2.3 - ProductRef

```

@Data
public class ProductRef {
    private final Product product;
    private final int weight; //вага продукту в 0.1 грама
    private final Long packageId;
}

```

Категорія Страви – сутність належить до довідника наведено у лістингу 2.4.

### Лістинг 2.4 - DishCategory

```

@Data
public class DishCategory {
    private final long categoryId;
    private String name;
}

```

Страва – сутність може бути довідниковою, або належати трапезі, чи дню наведені у лістингу 2.5.

### Лістинг 2.5 - Dish

```
@Data
public class Dish {
    private final long dishId;
    private String name;
    private String description;
    private DishCategory category;
    private final boolean template;
    private List<ProductRef> products;
}
```

Тип Трапези – довідникова сутність наведена у лістингу 2.6.

### Лістинг 2.6 - MealType

```
@Data
public class MealType {
    private final long typeId;
    private String name;
}
```

Трапеза – сутність яка містить Продукти і Страви наведені у лістингу 2.7.

### Лістинг 2.7 - Meal

```
@Data
public class Meal{
    private final long mealId;
    private String description;
    private MealType type;
    private List<Dish> dishes;
    private List<ProductRef> products;
}
```

День – сутність яка містить Трапези, Страви і Продукти приведені у лістингу 2.8.

#### Лістинг 2.8 - PlanDay

```
@Data
public class PlanDay{
    private final long dayId;
    private LocalDate date;
    private String description;
    private List<Meal> meals;
    private List<Dish> dishes;
    private List<ProductRef> products;
}
```

Турист – сутність, учасник, для якого розраховується раціон, для якого розподіляються пакунки наведених у лістингу 2.9.

#### Лістинг 2.9 - Hiker

```
@Data
public class Hiker {
    private final long id;
    private String name;
    private String description;
    private float weightCoefficient; //вагове навантаження
}
```

Пакунок – сутність яка об'єднує продукти для транспортування і розподілення наведених у лістингу 2.10.

#### Лістинг 2.10 - FoodPackage

```
@Data
public class FoodPackage {
    private final long id;
```

```

    private String name;
    private String description;
    private float volumeCoefficient; //об'ємний коефіцієнт
    private int additionalWeight;//вага пакування в 0.1 грам
}

```

Розкладка – центральна сутність системи наведена у лістингу 2.11.

### Лістинг 2.11 - FoodPlan

```

@Data
public class FoodPlan {
    private final long id;
    private String name;
    private String description;
    private ZonedDateTime createdOn;
    private ZonedDateTime lastUpdated;
    private List<PlanDay> days;
    private List<Hiker> members;
}

```

### 2.5.2 Схема бази даних

Серед великої кількості СКБД була обрана PostgreSQL. Це одна з найпотужніших і найгнучкіших реляційних баз даних із відкритим вихідним кодом, що робить її популярним вибором для розробників та підприємств. Її головною перевагою є високий рівень відповідності стандартам SQL, а також підтримка складних типів даних, таких як JSON, XML, геопросторові дані (через PostGIS), що дозволяє легко інтегрувати PostgreSQL у сучасні додатки. PostgreSQL також пропонує розширені можливості для роботи з транзакціями, такі як ACID-комплаєнс і MVCC (мультиверсійний контроль паралелізму), що гарантують цілісність і надійність даних навіть за умов високих навантажень (рис 2.4).

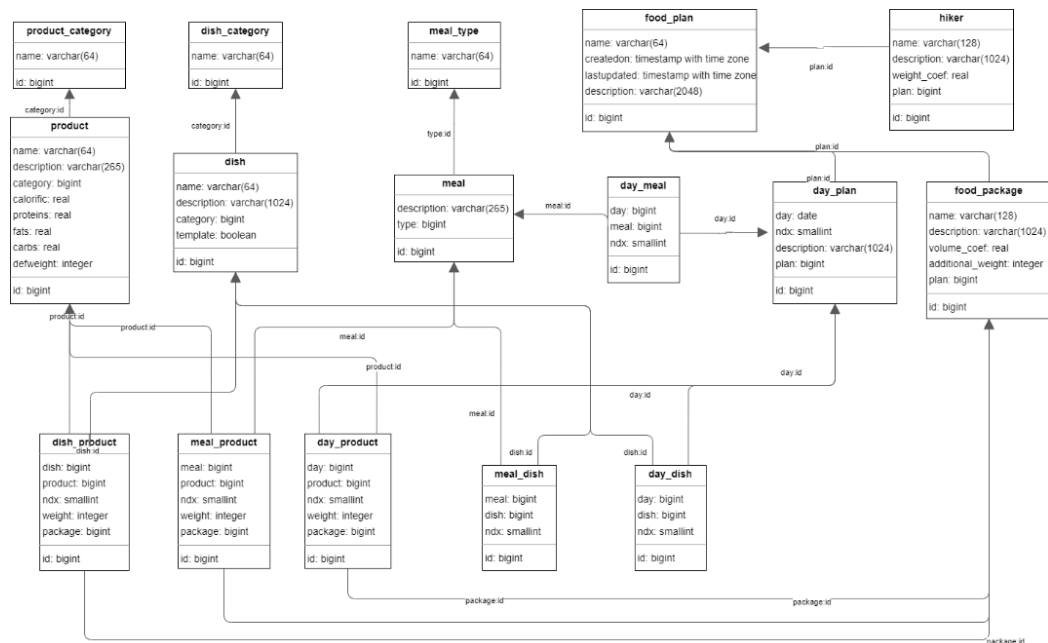


Рисунок 2.4 - структура бази даних

## 2.6 Проектування архітектури системи

Проектування системи, що складається з репозиторію для зберігання розкладок, серверної частини для складних обчислень і клієнтського веб-інтерфейсу для відображення і редагування розкладок, базується на чіткому поділі обов'язків між її компонентами. Репозиторій виконує функцію зберігання даних, забезпечуючи структуроване та цілісне збереження розкладок і пов'язаних сутностей, таких як продукти, учасники, дні та трапези. Використання реляційної бази даних, наприклад PostgreSQL, дозволяє створити ефективну модель даних із застосуванням нормалізації для забезпечення гнучкості та ефективності зберігання. Репозиторій також має забезпечувати можливість швидкого доступу до даних для серверної частини через чітко визначені запити.

Серверна частина системи виконує основну бізнес-логіку, зокрема, складні обчислення, такі як автоматичний розподіл продуктів між учасниками, генерація структурованих даних для розкладок і експорт у формати, зручні для подальшого

використання (наприклад, Excel). Реалізація серверної частини на платформі Spring Boot забезпечує високу продуктивність, модульність і підтримку сучасних стандартів розробки API. Цей компонент також слугує посередником між репозиторієм і клієнтським інтерфейсом, оброблюючи запити та забезпечуючи швидку й коректну передачу даних.

Клієнтський веб-інтерфейс виступає засобом взаємодії користувачів із системою, забезпечуючи зручне відображення й редагування розкладок та пов'язаних із ними даних. Використання сучасних веб-фреймворків, таких як VueJS, дозволяє створити адаптивний та інтерактивний інтерфейс. У ньому мають бути реалізовані функції перегляду, редагування й управління розкладками, а також забезпечена інтеграція з серверною частиною через REST API. Такий підхід забезпечує гнучкість системи, її масштабованість і зручність використання як на настільних пристроях, так і на мобільних платформах.

### **2.6.1 Проектування репозиторія системи**

Вибір системи зберігання даних — є важливим етапом проектування сучасних інформаційних систем і залежить від специфіки завдань, які система має вирішувати. SQL-бази, такі як PostgreSQL, забезпечують структуроване зберігання даних із чітко визначеними схемами, що робить їх ідеальними для додатків із складними запитамі, транзакціями та вимогами до цілісності даних. У той же час NoSQL-системи, такі як MongoDB або Cassandra, пропонують гнучкість у роботі з нереляційними даними, особливо у випадках із великими обсягами неструктурованої інформації, наприклад, логами або соціальними графами. Правильний вибір підходу залежить від архітектурних пріоритетів системи, зокрема швидкості обробки даних, необхідності масштабування та підтримки певної структури. Контейнеризація, як у випадку PostgreSQL у Docker, додає новий рівень універсальності, дозволяючи ефективно інтегрувати обраний тип бази даних у будь-яку архітектуру.

Розгортання бази даних PostgreSQL у Docker-контейнері має низку значних переваг, які роблять цей підхід популярним у сучасній розробці та експлуатації систем. Docker забезпечує ізольоване середовище для роботи

PostgreSQL, що мінімізує залежність від конфігурації хост-системи. Це дозволяє легко розгортати, оновлювати й переносити базу даних між різними середовищами (локальне середовище розробника, тестові сервери, продуктивна інфраструктура), зберігаючи ідентичність конфігурацій. Крім того, використання контейнерів значно спрощує управління версіями PostgreSQL, дозволяючи легко тестувати нові релізи без ризику для основної системи.

Ще однією перевагою є можливість інтеграції PostgreSQL у складніші архітектурні рішення. Контейнеризація дозволяє включити базу даних у контейнер на одному рівні з мікросервісами та оркеструвати її роботу за допомогою Kubernetes, забезпечуючи автоматичне масштабування, моніторинг і резервування. Також Docker дозволяє швидко налаштувати середовище з використанням стандартних інструментів, таких як Docker Compose, що спрощує інтеграцію бази даних із іншими компонентами системи, зокрема серверною частиною та сервісами кешування.

## **2.6.2 Проектування серверної частини**

Серверна частина використовує Domain Driven Design (DDD) — це підхід до проектування, який спрямований на чітке відображення бізнес-логіки в структурі програми. Він передбачає розподіл серверної частини на кілька шарів із центральною роллю доменної моделі. У порівнянні зі стандартною архітектурою (репозиторій → сервіс → контролер), DDD додає шар доменних сервісів і доменних об'єктів, які чітко відображають сутності бізнесу та їх взаємодію. Це забезпечує кращу модульність системи, дозволяє уникнути дублювання коду та спрощує підтримку навіть у випадках складної бізнес-логіки.

Основною перевагою DDD є створення автономного доменного шару, який не залежить від інфраструктурних деталей, таких як бази даних чи API. Цей підхід дозволяє мінімізувати зв'язність між бізнес-логікою та реалізацією її зберігання або взаємодії з іншими компонентами. Завдяки цьому репозиторій, відповідальний за доступ до даних, можна легко замінити, наприклад, із SQL на NoSQL, без змін у доменній логіці. Це робить систему гнучкою до майбутніх

змін або міграцій, а також забезпечує її незалежність від конкретної технології зберігання даних.

DDD дозволяє впроваджувати додатковий шар доменних сервісів, які виконують складні операції, що не належать до окремих сутностей. Наприклад, розрахунок ефективного розподілу ресурсів між групами користувачів можна помістити саме в доменний сервіс, що дозволяє чітко ізолювати цю логіку від інфраструктури та спростити тестування.

Обравши використання DDD, наступним важливим рішенням є вибір платформи для створення серверної частини. Серед багатьох альтернатив було обрано Spring Boot, який є ідеальним вибором для створення серверної частини, оскільки він забезпечує потужний набір інструментів для швидкого створення, розгортання та підтримки веб-додатків. Його перевага полягає в автоматичній конфігурації компонентів, гнучкій системі залежностей і підтримці багатьох популярних бібліотек. Завдяки Spring Boot можна легко реалізувати REST API, інтегрувати бази даних, працювати з безпекою через Spring Security і обробляти асинхронні процеси. Його екосистема добре інтегрується з DDD, дозволяючи створювати чітко структуровані програми.

Однією з ключових переваг Spring Boot є підтримка SpringJdbcTemplate, який надає зручний і продуктивний спосіб роботи з SQL-запитами. На відміну від ORM, таких як Hibernate, SpringJdbcTemplate дозволяє повністю контролювати виконання SQL-запитів, що є особливо корисним у системах із високими вимогами до продуктивності. SpringJdbcTemplate дозволяє створювати точні й ефективні SQL-запити, уникнувши небажаних трансформацій або генерації зайвого SQL-коду, як це може бути у Hibernate. Також він простіший у налаштуванні та не потребує попереднього визначення складних схем об'єктно-реляційного мапінгу (ORM).

Hibernate, у свою чергу, автоматизує багато рутинних процесів роботи з базою даних, таких як побудова CRUD-запитів, але ця автоматизація має свою ціну. Зокрема, вона може спричиняти перевантаження бази даних через непередумані запити або проблеми з оптимізацією, особливо у випадках із

великими наборами даних. Використання `SpringJdbcTemplate` усуває ці ризики, дозволяючи розробникам налаштовувати роботу з базою даних відповідно до потреб проєкту.

### 2.6.3 Проектування клієнтського веб інтерфейсу

Архітектура клієнтського веб-інтерфейсу була спроектована з використанням `Vue.js`, технології, яка базується на принципах модульності, реактивності та розділення відповідальностей.

Проектування архітектури клієнтського веб-інтерфейсу на базі `Vue.js` передбачає застосування сучасних патернів і принципів, які забезпечують гнучкість, масштабованість і легкість підтримки. `Vue.js` є прогресивним фреймворком, що дозволяє ефективно створювати інтерактивні інтерфейси та адаптувати їх під різні пристрої.

Основний принцип `Vue.js` — побудова інтерфейсу з невеликих, повторно використовуваних компонентів. Кожен компонент відповідає за певну функцію або частину інтерфейсу, наприклад, форму, таблицю або кнопку. Цей підхід спрощує тестування, підтримку та розширення системи. Важливим є чітке визначення взаємодії між компонентами через властивості (`props`) і події (`events`).

Використання принципу розділення відповідальностей у `Vue.js` допомагає відокремлювати логіку бізнесу, представлення й обробку даних. Для цього застосовуються директиви (`v-bind`, `v-model`) і бібліотеки для управління станом, такі як `Vuex` або `Pinia`, що централізують управління станом додатка.

У `Vue.js` компоненти підтримують односторонній потік даних (з батьківського до дочірнього компонента), що зменшує ймовірність помилок і забезпечує передбачувану поведінку інтерфейсу. Для передачі змін у зворотному напрямку використовуються події, що відповідає принципу "підйому стану" (`state lifting`).

У процесі проектування клієнтського інтерфейсу з `Vue.js` важливо використовувати принципи адаптивного дизайну для забезпечення сумісності з різними пристроями. CSS-фреймворки, такі як `Bootstrap`, допомагають швидко створювати сучасні інтерфейси.

## 2.7 Розробка клієнтського веб інтерфейсу

Розробка клієнтського веб-інтерфейсу з використанням Vue.js базується на сучасних інструментах розробки, таких як Node.js і npm. Node.js забезпечує середовище виконання JavaScript поза браузером, яке дозволяє виконувати серверну логіку, керувати залежностями та створювати клієнтські додатки. NPM (Node Package Manager) використовується для управління бібліотеками, плагінами та іншими компонентами, що прискорюють розробку.

Стандартна структура проєкту на Vue.js із використанням Node.js та npm виглядає так:

- а) `node_modules/`: Залежності, встановлені через npm;
- б) `public/`: Статичні файли (HTML, зображення, favicon тощо);
- в) `src/`: Основний код додатка:
  - 1) `assets/`: Статичні ресурси (зображення, стилі);
  - 2) `components/`: Повторно використовувані Vue-компоненти;
  - 3) `views/`: Веб-сторінки (або основні складові всіх маршрутів);
  - 4) `router/`: Налаштування маршрутизації (Vue Router);
  - 5) `store/`: Централізоване управління станом (Vuex/Pinia);
  - 6) `App.vue`: Головний компонент додатка;
  - 7) `main.js`: Точка входу, що ініціалізує Vue-додаток.
- г) `package.json`: Конфігурація проєкту, що включає скрипти npm та залежності;
- д) `vue.config.js`: Додаткові налаштування Webpack або Vite;
- е) `tests/`: Тести (юніт-тести, e2e-тести).

Головними етапами розробки є встановлення необхідних залежностей через npm, таких як Vue Router, Axios, тощо.

Наступним кроком є реалізація компонента `App.vue` для побудови основного каркаса додатка (хедер, футер, навігація).

Налаштування маршрутизації в проєктах на основі Vue.js виконується за допомогою бібліотеки Vue Router, яка забезпечує зручний спосіб організації навігації між компонентами. У файлі маршрутизації (`router/index.js`, дивись Лістинг A.1) створюється об'єкт маршрутизатора за допомогою `createRouter`, де визначаються всі маршрути програми. Кожен маршрут має обов'язкові параметри, такі як шлях (`path`) і компонент (`component`), який повинен бути відображений при переході на цей шлях. Додатково можна налаштувати іменовані маршрути, динамічні параметри (наприклад, `/plan/:id`), редиректи та захист маршруту через "guard" функції.

Після налаштування маршрутизатор підключається до Vue-додатку у файлі `main.js`, де його додають як плагін через `app.use(router)`. Це дозволяє використовувати Vue Router для навігації через директиву `<router-link>` або програмно через методи, такі як `router.push()`. Такий підхід робить маршрутизацію гнучкою, дозволяючи легко додавати нові сторінки, забезпечувати захист доступу до певних маршрутів, а також динамічно змінювати інтерфейс залежно від активного маршруту.

Важливим при створенні компонентів є інкапсуляція логіки та стилів у компоненти для спрощення тестування та розширення. Реалізація взаємодії з сервером відбувається через `axios`. Для завантаження даних використовуються Vue lifecycle hooks (`mounted`, `created`). Нижче наведені два компоненти, які реалізують секцію учасників туристичної подорожі `HikerView` та кожен окремий рядок таблиці `HikerItem` (Лістинги A.2, A.3).

## 2.8 Розробка серверної частини

Розробка серверної частини на основі Spring Boot із використанням Maven дозволяє створювати модульну, гнучку та масштабовану систему, що відповідає принципам Domain Driven Design (DDD). Maven є потужним інструментом для

управління проєктами, який надає можливість організувати систему у вигляді багатомодульної структури. Кожен модуль виконує свою чітко визначену роль у загальній архітектурі та може розроблятися, тестуватися і підтримуватися окремо. Наприклад, модулі `domain-tier` (рівень доменних сервісів), `web-tier` (рівень бізнес логіки та представлення), та `repository` (інфраструктурнозалежний рівень репозиторія) розташовані у спільному кореневому проєкті, кожен із власним `pom.xml`, що визначає його залежності та конфігурацію. Головний модуль (`root`) управляє загальними залежностями, налаштуваннями збірки та інтеграцією між усіма підпроєктами, забезпечуючи координацію роботи системи. Такий підхід дозволяє зберігати чіткість структури, зменшує взаємозалежність між частинами системи і значно полегшує їхнє розширення чи модифікацію.

В DDD архітектура системи організована навколо домену (області предметної діяльності). Розглянемо кожен модуль серверної частини починаючи з доменного:

Перший Модуль `domain-tier` (доменний рівень)

Цей модуль є центральним у DDD і містить:

- доменні моделі — класи, що представляють сутності (Entities, наприклад `Hiker`) та об'єкти значення (Value Objects) у системі наведені у лістингу 2.12.

#### Лістинг 2.12 - `Hiker`

```
@Data
public class Hiker {
    private final long id;
    private String name;
    private String description;
    private float weightCoefficient;
}
```

- доменні сервіси — класи, що реалізують бізнес-логіку, яка не належить до конкретної сутності, як наприклад `HikerDomainService`, дивись Лістинг А.4;

- репозиторії (інтерфейси) — визначення методів доступу до даних, які не залежать від типу бази даних чи інфраструктури, наприклад `IhikerRepo` наведені у лістингу 2.13.

### Лістинг 2.13 - `IhikerRepo`

```
public interface IhikerRepo {
    List<Hiker> getPlanHikers(long planId);
    Optional<Hiker> getHiker(long id);
    long addHiker(long planId, Hiker hiker);
    boolean updateHiker(Hiker hiker);
    void deleteHiker(long id);
    boolean existsHiker(long id);
}
```

Другий модуль `repository` (інфраструктурнозалежний рівень репозиторія)

Цей модуль містить:

- реалізації репозиторіїв з використанням `SpringJdbcTemplate`, наприклад `HikerRepo`, дивись Лістинг А.5;
- налаштування `Spring Boot` для роботи з базою даних, безпекою, кешуванням тощо наведено у лістингу 2.14.

### Лістинг 2.14 - `RepositoryConfig`

```
@Configuration
@ComponentScan(basePackages="com.outdoor.foodcalc")
@PropertySource("classpath:database.properties")
public class RepositoryConfig {

    @Value("${jdbc.url}")
    private String url;
```

```

    @Value("${jdbc.user}")
    private String user;
    @Value("${jdbc.password}")
    private String password;
    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new
DriverManagerDataSource();
        dataSource.setDriverClassName("org.postgresql.Driver");
        dataSource.setUrl(url);
        dataSource.setUsername(user);
        dataSource.setPassword(password);
        return dataSource;
    }
}

```

Клас `RepositoryConfig` створює і налаштовує джерело даних, яке Spring використовуватиме для підключення до бази даних PostgreSQL. Це джерело даних може бути передано в інші компоненти, наприклад, для роботи з JDBC. Файл `database.properties`, який згаданий у `@PropertySource`, має містити такі властивості: `jdbc.url`, `jdbc.user`, `jdbc.password`. Поля `url`, `user` та `password` позначені анотацією `@Value`, що дозволяє Spring автоматично підставляти значення зі згаданого файлу властивостей, наприклад `@Value("${jdbc.url}")` — завантажує значення властивості `jdbc.url` (URL для підключення до бази даних). Це забезпечує можливість легко змінювати конфігурацію бази даних без необхідності змін у коді. Такий підхід дозволяє легко змінювати параметри підключення або адаптувати програму до різних середовищ, змінюючи лише файл властивостей.

Третій модуль `web-tier` (рівень бізнес логіки та представлення)

У цьому модулі реалізуються виклики до доменних сервісів, підготовка даних для відправлення клієнту через REST або інших клієнтів:

- обробка вхідних даних і формування відповіді;

- валідація запитів;
- реалізація REST-контролерів для обробки HTTP-запитів, наприклад HikerEndpoint, дивись Лістинг А.6.

Окрім типових ендпоінтів \ сервісів \ репозиторіїв треба зазначити важливу функціональність реалізовану за допомогою бібліотеки Apache POI – експорт даних в зручному Excel форматі. Apache POI дозволяє створювати структуровані, читабельні таблиці, які легко переглядати, редагувати та обмінюватися ними з іншими учасниками. У форматі Excel дані зручно сортувати, фільтрувати та використовувати для подальших розрахунків, що особливо корисно для планування багатоденних туристичних походів.

Однією з головних переваг такого експорту є можливість автоматизації створення звітів із детальною інформацією. Наприклад, розкладка по днях може включати страви, продукти та їхню вагу, що дозволяє чітко планувати харчування на кожен день. Списки продуктів та їх розподіл по пакунках, що представлені у вигляді окремих таблиць, значно полегшують підготовку і транспортування. Завдяки інтеграції Apache POI у серверну частину можна динамічно генерувати такі звіти в реальному часі, забезпечуючи користувачів актуальними та точними даними.

Ще однією значною перевагою експорту в Excel за допомогою Apache POI є можливість використовувати цю функцію безпосередньо через API, без необхідності роботи з веб-інтерфейсом. Серверна частина може обробляти HTTP GET-запити, генеруючи Excel-файли на льоту та передаючи їх у відповідь. Наприклад, користувач може зробити запит через інструмент для тестування API, такий як Postman, і отримати сформований файл Excel безпосередньо у відповідь. Це забезпечує гнучкість у використанні функціоналу експорту, дозволяючи інтегрувати його у зовнішні системи, автоматизовані сценарії чи використовувати в розподілених середовищах без необхідності взаємодії з веб-додатком. Такий підхід спрощує доступ до даних і робить систему більш універсальною.

Крім того, результати пошуку ефективного розподілення продуктів за допомогою алгоритмів оптимізації, виведені в Excel, дають змогу швидко оцінити планування та внести необхідні корективи. Завдяки можливостям форматування Excel, такі як кольорові позначки, підсумкові рядки та графіки, звіти стають ще зрозумілішими та інформативнішими. Apache POI забезпечує високий рівень контролю над створенням і стилізацією документів, що робить цей інструмент ідеальним для розробників, які прагнуть забезпечити користувачів зручними і наочними звітами.

Експорт був реалізований для наступних даних (таб 2.1):

- структурована розкладка по днях \ трапезах;
- загальний список продуктів туристичної подорожі;
- список пакунків з продуктами, які до них належать;
- ефективне розподілення продуктів серед учасників (кожен учасник на окремому аркуші).

Таблиця 2.1 – експорт розкладки по днях експортований в Excel

|                                     |           |        |       |            |      |
|-------------------------------------|-----------|--------|-------|------------|------|
| Назва:                              | Тянь-Шань |        |       |            |      |
| Учасники                            | 5         |        |       |            |      |
| Меню похода по днях:                |           |        |       |            |      |
| Назва                               | ККал      | Білків | Жирів | Вуглеводів | Вага |
| Меню на 2025-01-20                  | 2410      | 54.4   | 36.25 | 395.35     | 1850 |
| День 1. Заброска у долину Кизил-Суу |           |        |       |            |      |
| Обід                                | 751       | 15.5   | 11.15 | 118.35     | 805  |
| компот з солодощами                 | 102       | 0.15   | 0.35  | 25.2       | 225  |
| Ягоди сублімовані                   | 4.5       | 0      | 0     | 1          | 75   |
| Цукор                               | 80        | 0      | 0     | 19.95      | 100  |
| Цукерки                             | 17.5      | 0.15   | 0.35  | 4.25       | 50   |

|                                     |           |        |       |            |      |
|-------------------------------------|-----------|--------|-------|------------|------|
| Продовження таблиці 2.1             |           |        |       |            |      |
| Назва:                              | Тянь-Шань |        |       |            |      |
| Учасники                            | 5         |        |       |            |      |
| Меню похода по днях:                |           |        |       |            |      |
| Назва                               | ККал      | Білків | Жирів | Вуглеводів | Вага |
| Меню на 2025-01-20                  | 2410      | 54.4   | 36.25 | 395.35     | 1850 |
| День 1. Заброска у долину Кизил-Суу |           |        |       |            |      |
| рисовий суп з м'ясом                | 314       | 8.1    | 10.2  | 23         | 330  |
| рис                                 | 67        | 1.35   | 0.2   | 14.55      | 100  |
| картопля сушена                     | 35.25     | 0.7    | 0     | 8.15       | 75   |
| Мясо сублімоване                    | 210       | 6      | 10    | 0          | 100  |
| Морква сушена                       | 0.5       | 0.05   | 0     | 0.1        | 10   |
| Цибуля ріпчаста сушена              | 1.2       | 0.05   | 0     | 0.2        | 15   |
| Перець солодкий сушений             | 0.05      | 0      | 0     | 0          | 5    |
| Сіль                                | 0         | 0      | 0     | 0          | 15   |
| Перець чорний                       | 0         | 0      | 0     | 0          | 5    |
| Зелень сушена                       | 0         | 0      | 0     | 0          | 5    |
| Перекус                             | 362.5     | 6.25   | 0     | 84.4       | 250  |
| Вечеря                              | 1296      | 32.65  | 25.1  | 192.6      | 795  |
| Гречана каша з м'ясом               | 1146      | 32.15  | 23.4  | 155.65     | 575  |
| Каша гречана                        | 777       | 21.55  | 5.9   | 155.35     | 350  |
| Мясо сублімоване                    | 367.5     | 10.5   | 17.5  | 0          | 175  |
| Цибуля ріпчаста сушена              | 1.2       | 0.05   | 0     | 0.2        | 15   |
| Морква сушена                       | 0.5       | 0.05   | 0     | 0.1        | 10   |
| Сіль                                | 0         | 0      | 0     | 0          | 15   |
| Перець чорний                       | 0         | 0      | 0     | 0          | 5    |
| Зелень сушена                       | 0         | 0      | 0     | 0          | 5    |

## 2.9 Алгоритм пошуку ефективного розподілення

Головна мета розробки системи – автоматизація пошуку ефективного розподілення продуктів. Пошук такого розподілення викликаний необхідністю ефективнішого розподілення продуктів, обладнання або інших ресурсів серед учасників групи, щоб мінімізувати фізичне навантаження і врахувати інші обмеження, наприклад, індивідуальні можливості чи специфіку маршрутів. Це завдання стає особливо складним, коли кількість варіантів розподілу (комбінацій) стрімко зростає залежно від числа учасників і пакунків, що потребує використання комбінаторних алгоритмів.

Для задачі розподілення  $M$  пакунків серед  $N$  учасників туристичної подорожі кількість можливих комбінацій дорівнює  $N^M$ . Наприклад, якщо є 10 продуктів і 5 учасників, кількість можливих розподілів складе дорівнює  $5^{10} = 9,765,625$ . Такий експоненційний ріст числа комбінацій значно ускладнює перебір усіх можливих варіантів, особливо для великих значень  $M$  і  $N$ . Це призводить до потреби в алгоритмах оптимізації, які можуть знайти найкращий варіант розподілу без повного перебору.

Важливу роль у зменшенні кількості можливих комбінацій відіграють пакунки, які об'єднують довільні набори продуктів у єдину сутність. Замість розгляду кожного продукту окремо, використання пакунків дозволяє суттєво знизити кількість одиниць, що підлягають розподілу, й відповідно скоротити кількість комбінацій. Наприклад, якщо 50 продуктів об'єднати в 10 пакунків, задача розподілення перетворюється на оперування меншою кількістю елементів, що спрощує обчислення і зменшує експоненціальний ріст комбінацій. Це також дозволяє ефективніше організовувати транспортування та зберігання продуктів.

Оцінюючи пакунки для ефективного розподілення серед учасників, недостатньо брати до уваги лише вагу як ключовий критерій. Вага є важливим, але не єдиним фактором, що впливає на рівномірність розподілу та зручність

транспортування. Об'єм пакунку також грає вирішальну роль, оскільки він визначає фізичний простір, необхідний для розміщення вантажу, а також впливає на балансування рюкзаків або інших транспортних засобів.

Наприклад, пакунок із сушеними овочами для супів вага якого 400 грам (рис 2.5) займає значно більше об'єму, ніж пакунок із тушонкою такої ж ваги (рис 2.6). Тушонка, будучи компактною, може бути зручно розміщена в рюкзаку без значного впливу на загальний обсяг, тоді як об'ємний пакунок із сушеними овочами створить дисбаланс і обмежить простір для інших необхідних речей. Тому об'єм і форма пакунку повинні враховуватися в моделі оцінки, щоб забезпечити справедливий і практичний розподіл ресурсів.



Рисунок 2.5 - Пакунок з набору сушених овочів на 3 супи вагою ~400 гр



Рисунок 2.6 - Пакунок упаковок тушонки вагою ~1000 гр

У задачах розподілення ваги та об'єму між учасниками подорожі використовуються різні оціночні функції для визначення якості розподілу. Найпростішими є функції, що враховують лише сумарну вагу вантажів, закріплених за кожним учасником. Проте такі підходи ігнорують об'єм пакунків, який може суттєво впливати на зручність транспортування. Більш комплексними є функції, які враховують вагу, об'єм або їх комбінацію. Серед цих підходів найкращим є використання середньоквадратичного відхилення, розширеного об'ємним коефіцієнтом. Цей метод забезпечує точнішу оцінку рівномірності розподілення, дозволяючи враховувати як фізичну вагу, так і об'єм пакунків.

Для об'єктивної оцінки розподілення використовується скоригована метрика, яка поєднує вагу ( $w_i$ ) та об'єм ( $v_i$ ) кожного пакунку через об'ємний коефіцієнт ( $k$ ) за формулою 2.1:

$$W_i = w_i + k * v_i \quad (2.1)$$

де  $k$  — коефіцієнт, який визначає вагомість об'єму у загальній оцінці. Якщо простір для транспортування обмежений (наприклад, рюкзаки або човни), значення  $k$  збільшується, щоб відобразити важливість оптимізації об'єму. Цей підхід дозволяє враховувати реальні фізичні характеристики пакунків, наприклад, пакунок із сушеними овочами вагою 400 грам матиме більший вплив через свій об'єм, ніж пакунок тушонки тієї ж ваги.

Після обчислення скоригованої ваги ( $W_i$ ) для кожного пакунку визначається сумарна зважена вага ( $S_j$ ) для кожного учасника за формулою 2.2:

$$S_j = \sum_{i \in P_j} W_i \quad (2.2)$$

де  $P_j$  — множина пакунків, закріплених за учасником  $j$

Середня вага на одного учасника за формулою 2.3:

$$\bar{S} = \frac{\sum_{j=1}^N S_j}{N} \quad (2.3)$$

Середньоквадратичне відхилення, яке показує рівномірність розподілу за формулою 2.4:

$$\sigma = \sqrt{\frac{\sum_{j=1}^N (S_j - \bar{S})^2}{N}} \quad (2.4)$$

Чим менше значення  $\sigma$ , тим рівномірніше розподілені вантажі між учасниками з урахуванням як ваги, так і об'єму. Високе значення  $\sigma$  вказує на значні відхилення від середнього, що може свідчити про дисбаланс у розподілі. Наприклад, один учасник може отримати пакунки з великою масою і об'ємом, тоді як інший — легкі та компактні.

Використання середньоквадратичного відхилення як оціночної функції дозволяє не лише аналізувати якість розподілення, але й застосовувати його як цільову функцію в оптимізаційних алгоритмах. Алгоритм мінімізує  $\sigma$ , забезпечуючи рівномірність навантаження. Додавання об'ємного коефіцієнта дає змогу враховувати специфічні обмеження простору та форми вантажу. Такий підхід дозволяє забезпечити баланс між рівномірністю розподілу та практичністю транспортування, підвищуючи комфорт учасників і ефективність планування.

Визначивши оціночну функцію, можна перейти до фільтрації заздалегідь неефективних розподілень на етапі генерації, що є критично важливим кроком в алгоритмах оптимізації, особливо при роботі з комбінаторними задачами, такими як ефективний розподіл пакунків серед учасників. Враховуючи експоненційний ріст кількості можливих комбінацій, фільтрація дозволяє значно

скоротити простір пошуку, зосередившись лише на потенційно корисних варіантах. Це не тільки прискорює обчислення, але й дозволяє ефективніше використовувати ресурси системи.

На етапі генерації кожної комбінації можна застосовувати правила відсікання, які виключають явно некоректні або нераціональні розподілення. Наприклад, комбінації, де навантаження на одного учасника суттєво перевищує середнє значення або коли окремий учасник отримує всі об'ємні пакунки, можуть бути відхилені без подальшого аналізу. Це забезпечує, що алгоритм працює лише з тими варіантами, які мають реальний шанс стати найефективнішими. Це не тільки знижує час виконання алгоритму, але й зменшує навантаження на подальші етапи обчислень, такі як оцінювання за допомогою середньоквадратичного відхилення чи інших метрик.

Застосування фільтрації на етапі генерації розподілення дозволяє більш ефективно використовувати алгоритмічні ресурси, забезпечуючи баланс між точністю та швидкістю обчислень. Такий підхід є невід'ємною частиною оптимізаційних алгоритмів, які мають працювати в реальних умовах із великими обсягами даних і жорсткими обмеженнями.

Критерії ефективного розподілу продуктів:

- приблизно рівна вага продуктів у кожного учасника походу на початку походу;
- приблизно рівна вага продуктів у кожного учасника походу протягом кожного дня;
- вага продуктів кожного учасника протягом всіх днів має бути монотонно спадаючою;
- фізичний об'єм продуктів має бути приблизно рівним.

Задоволення цих критеріїв базується на мінімізації значення оціночної функції (перший, другий і четвертий критерії), а також на впровадженні спеціальної стратегії пошуку (третій критерій).

Якщо здійснювати пошук розподілення продуктів у хронологічному порядку (від першого до останнього дня), виникає ризик, що обрані ефективні

розподілення на початку створюють нерівномірність у наступних днях. Для задоволення критерію монотонності пошук розподілення починається з останнього дня і рухається в зворотному порядку до першого. Цей підхід має такі переваги:

- гарантія виконання критерію. Починаючи з останнього дня, алгоритм визначає остаточний розподіл ваги для кожного учасника на завершення подорожі. Цей розподіл стає базою для розрахунків попередніх днів, що дозволяє забезпечити поступове зменшення ваги;
- обмеження варіантів. Розподіл для кожного попереднього дня повинен враховувати вагу, яка залишиться на наступний день. Це значно зменшує кількість можливих комбінацій для розгляду, що прискорює пошук;
- контроль зменшення ваги. Для кожного учасника на попередньому дні вагу продуктів додають так, щоб їх сумарна вага разом із залишком на наступний день не перевищувала допустимої норми і водночас забезпечувала поступове зменшення.

Особливістю розподілу продуктів у туристичній подорожі є неможливість досягнення абсолютно рівного навантаження для всіх учасників. Це зумовлено різними характеристиками продуктів, такими як вага, об'єм, а також специфікою пакунків. Щоб уникнути надмірно тривалого пошуку ідеального розподілення, необхідно впровадити поняття допустимої похибки — максимальної різниці між вагою (або об'ємом) продуктів, розподілених між учасниками, яка вважається прийнятною. Застосування похибки надає наступні переваги:

- прискорення пошуку. Допустима похибка дозволяє уникнути опрацювання варіантів, які намагаються досягти мінімального покращення, що не має суттєвого впливу на загальну якість розподілу. Наприклад, якщо різниця у вазі між учасниками знаходиться в межах 5% від середньої, подальша оптимізація стає недоцільною;
- зменшення комбінаторної складності. Замість перебору всіх можливих варіантів алгоритм зосереджується лише на тих, які знаходяться в межах похибки, що значно скорочує кількість комбінацій для аналізу;

– гнучкість і адаптивність. Похибка може бути налаштована залежно від особливостей подорожі. Наприклад, у коротких походах допустима похибка може бути меншою, адже навантаження не є критичним. У довгих походах із великими групами допустима похибка може бути більшою, щоб уникнути надмірних обчислень.

На основі всього вищезазначеного, можемо сформулювати наступний алгоритм ефективного розподілення продуктів у туристичній подорожі, який базується на алгоритмі пошуку в глибину в графі та враховує специфічні критерії розподілу ваги й об'єму продуктів між учасниками подорожі:

а) завантаження необхідних даних. Алгоритм починається з підготовки початкових даних. Завантажуються розкладка по днях, список учасників, продуктів і їх властивості;

б) генерація всіх можливих розподілень для останнього дня. На першому етапі генеруються всі можливі варіанти розподілу продуктів серед учасників для останнього дня подорожі. Цей етап важливий, оскільки третій критерій (монотонне зменшення ваги) вимагає, щоб пошук починався з кінця розкладки;

в) фільтрація явно неефективних розподілень. Згенеровані варіанти оцінюються за допомогою оціночної функції, яка враховує вагу, об'єм і допустиму похибку. Розподілення, які значно перевищують допустиму похибку або порушують інші критерії, відкидаються. Це суттєво зменшує кількість комбінацій для подальшого аналізу;

г) початок пошуку для передостаннього дня. Алгоритм переходить до аналізу розподілень для передостаннього дня. На цьому етапі здійснюються наступні дії:

- 1) вибір першого неопрацьованого розподілення для поточного дня;
- 2) якщо поточний день є першим. У випадку, коли поточний день є першим у розкладці, алгоритм виконує пошук найкращого розподілення серед усіх варіантів для цього дня. Найкраще розподілення визначається за

значенням оціночної функції, після чого всі розподілення позначаються як опрацьовані;

3) якщо це не перший день. Для днів, які передують останньому, алгоритм виконує наступні дії:

- генерує всі можливі розподілення для наступного дня ( $i-1$ );
- виконує фільтрацію явно неефективних розподілень;
- продовжує пошук у глибину для наступного дня ( $i-1$ ).

4) якщо всі розподілення опрацьовані. Якщо всі розподілення для поточного дня опрацьовані, алгоритм повертається до аналізу варіантів для попереднього дня.

д) Завершення пошуку. Алгоритм продовжує працювати, доки:

- 1) існують неопрацьовані розподілення для поточного дня;
- 2) оціночна функція найкращого розподілення не досягне цільового значення (мінімуму).

Як тільки всі можливості будуть вичерпані або знайдено ефективне рішення, алгоритм завершує роботу, видаючи найкраще розподілення продуктів (рис 2.7).

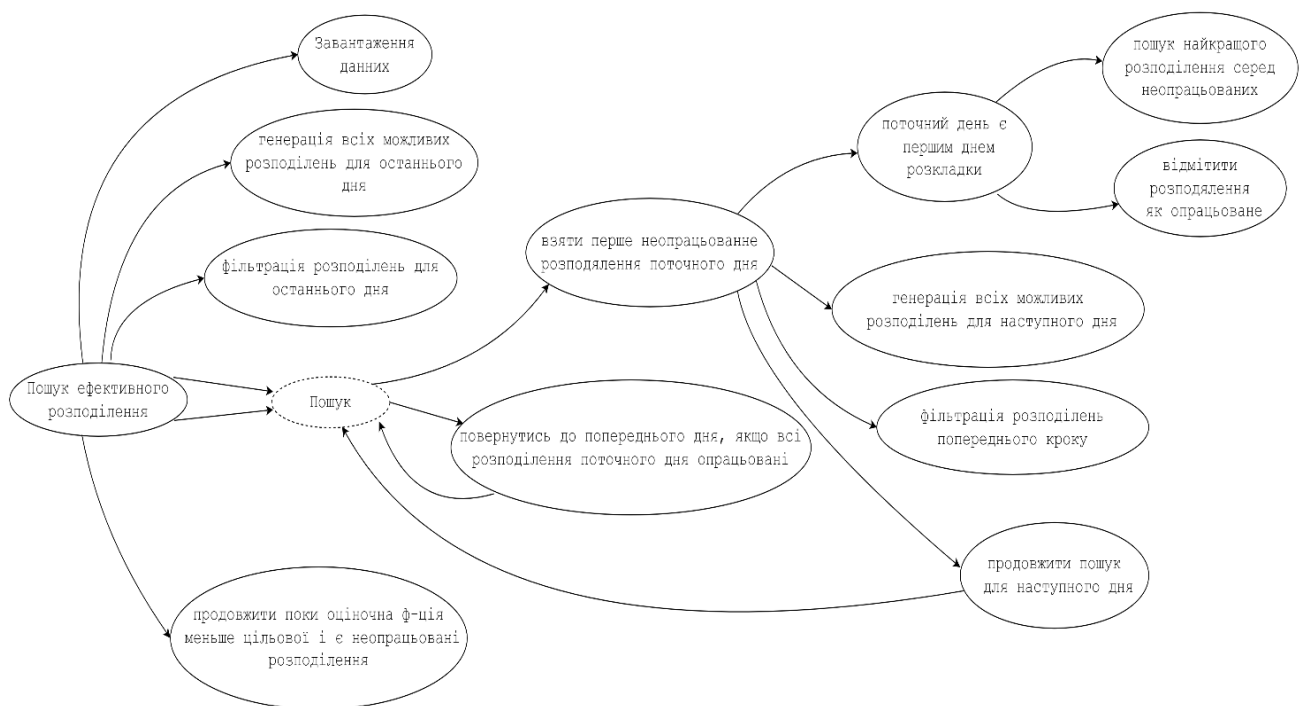


Рисунок 2.7 – алгоритм найкращого розподілу продуктів

Цей алгоритм забезпечує баланс між точністю та швидкістю обчислень. Основні переваги алгоритму:

- ефективність. Завдяки фільтрації значно зменшується кількість неефективних розподілень і допуск припустимої похибки що вагомо скорочує обсяг пошуку;
- врахування критеріїв. Алгоритм одночасно враховує вагу, об'єм, монотонність і рівномірність розподілу;
- модульність. Можливість налаштування окремих критеріїв та параметрів, таких як похибка, об'ємний коефіцієнт тощо;
- гнучкість. Пошук починається з останнього дня, що дозволяє краще задовольняти критерій монотонності.

Реалізація алгоритму знаходиться в класах `FoodDistributionEngine` та `FoodDistributionsStorage`. Вона наведена в Лістингах A.7 і A8.

## **2.10 Висновки до розробки системи моделювання ефективного розподілення ваги**

У другому розділі був здійснений комплексний аналіз і проєктування складної клієнт-серверної системи, яка включає інтеграцію бази даних, серверної частини для обчислень та веб-інтерфейсу для користувачів. Основною метою було створення системи, яка відповідає сучасним стандартам розробки, забезпечує високу продуктивність і масштабованість. У процесі роботи використовувалися найкращі практики програмної інженерії, що дозволило не лише реалізувати функціональні вимоги, але й закласти основу для подальшого розвитку й підтримки системи.

Особливу увагу було приділено визначенню технічних характеристик системи, які включають апаратні вимоги. Були проаналізовані потреби

клієнтської частини, серверної частини та бази даних, що дозволило визначити ефективну конфігурацію обладнання. Було запропоновано використання багатоядерних процесорів для забезпечення швидкої обробки запитів, достатнього обсягу оперативної пам'яті для стабільної роботи під час високих навантажень, а також високопродуктивних SSD для швидкого доступу до даних. Таке детальне опрацювання апаратних вимог забезпечує продуктивність і стабільність системи навіть за умов пікових навантажень.

Програмні вимоги були розроблені з урахуванням сучасних технологічних стандартів і вимог користувачів. Було запропоновано використання Vue.js для створення клієнтської частини, що забезпечує інтерактивність і адаптивність інтерфейсу, а також Spring Boot для серверної частини, яка забезпечує надійність і модульність системи. Для бази даних було обрано PostgreSQL, яка пропонує високу продуктивність і розширюваність. Такий підхід гарантує не лише якість реалізації поточних функцій, але й спрощує інтеграцію нових модулів у майбутньому.

Архітектура системи була спроектована із застосуванням Domain Driven Design (DDD), що дозволило чітко розмежувати бізнес-логіку, інфраструктуру та взаємодію з користувачем. Використання окремого доменного шару, а також інтеграція репозиторію й API забезпечило гнучкість системи. Таке проєктування дозволяє легко адаптувати систему до нових вимог, наприклад, замінити SQL-базу даних на NoSQL без необхідності змін у бізнес-логіці.

Клієнтський веб-інтерфейс був розроблений таким чином, щоб максимально спростити взаємодію користувачів із системою. Використання компонентно-орієнтованої архітектури в Vue.js дозволило створити модульний і повторно використовуваний код. Інтерфейс підтримує адаптивний дизайн, що забезпечує коректну роботу на різних пристроях, включаючи смартфони та планшети. Були враховані принципи доступності (WCAG), що робить систему інклюзивною для всіх категорій користувачів.

Серверна частина була реалізована з використанням Spring Boot, що забезпечує продуктивність і модульність, а також підтримує інтеграцію з

різними типами баз даних. Було реалізовано ефективний алгоритм розподілу ваги між елементами, який оптимізує обчислювальні процеси. Використання SpringJdbcTemplate замість ORM, такого як Hibernate, дозволило забезпечити більший контроль над SQL-запитами, що позитивно вплинуло на продуктивність системи.

У результаті виконаної роботи вдалося створити систему, яка відповідає сучасним вимогам до програмного забезпечення. Висока продуктивність, зручність використання, надійність і масштабованість роблять її готовою до впровадження в реальні умови експлуатації. Застосування найкращих практик розробки гарантує якість і тривалу підтримку системи. Це дозволяє ефективно вирішувати завдання користувачів і забезпечує гнучкість для подальших оновлень та вдосконалень.

### **3 ДОСЛІДЖЕННЯ СИСТЕМИ**

Випробування системи, розроблення якої представлено у другому розділі, є критично важливим етапом для підтвердження її працездатності та ефективності. Теоретичні припущення, закладені в основу системи, повинні бути перевірені на практиці, щоб оцінити їх відповідність реальним умовам використання. Це включає перевірку алгоритмів, критеріїв оцінювання, структурних компонентів системи, таких як серверна частина, клієнтський інтерфейс і бази даних. Випробування дозволяють не лише переконатися у правильності теоретичних розрахунків, а й виявити потенційні проблеми чи обмеження, які могли залишитися непоміченими на етапі проектування.

Проведення експериментів є ключовим етапом перевірки моделі, оскільки вони дозволяють оцінити поведінку системи в різних сценаріях. Наприклад, експерименти можуть включати тестування розподілення продуктів за різними вхідними даними, моделювання навантаження на сервер, а також перевірку

роботи алгоритмів оптимізації в умовах великого обсягу даних. Результати експериментів можуть підтвердити правильність вибору параметрів, таких як допустима похибка, або виявити необхідність їх коригування. Крім того, експерименти допомагають оцінити продуктивність системи, стабільність роботи і відповідність заявленим критеріям оптимізації.

Підтвердження правильності припущень через експерименти дозволяє впевнитися, що розроблена система відповідає поставленим цілям і може бути ефективно використана у реальних умовах. Успішне тестування моделі на різних наборах даних і сценаріях забезпечує довіру до її функціонування і створює основу для подальшого вдосконалення. Виявлення недоліків на етапі випробувань дозволяє внести необхідні корективи ще до впровадження системи, що знижує ризики і забезпечує її успішну експлуатацію.

### **3.1 Підготовка системи до досліджень**

Першим етапом підготовки системи є визначення необхідного обсягу даних для перевірки. Система розподілення ваги вирішує комбінаторну задачу, яку зручно перевірити на невеликих обсягах даних, таких як розподіл 9-10 пакунків між 3-4 учасниками. Це дозволяє ретельно перевірити правильність застосування ключових коефіцієнтів, зокрема коефіцієнта об'єму пакунка, який враховує фізичний простір, та метрики рівномірності розподілення ваги між учасниками. Оскільки кількість можливих комбінацій у таких випадках є обмеженою, їх легше перевірити вручну. Це дає змогу порівняти результати алгоритму з теоретично ефективними варіантами, щоб упевнитися у правильності реалізації критеріїв оцінювання.

Експеримент із невеликою кількістю учасників і пакунків також сприяє розумінню поведінки алгоритму на різних етапах: від генерації розподілень до застосування оціночних функцій і фільтрації неефективних варіантів. Це

дозволяє детально перевірити, чи враховує алгоритм усі необхідні фактори, наприклад, збалансованість ваги й об'єму пакунків для кожного учасника. Крім того, ручна перевірка результатів гарантує точність алгоритму та допомагає виявити можливі недоліки у використанні коефіцієнтів або оцінювальних метрик. Такий підхід забезпечує впевненість у коректності системи перед масштабуванням на великі набори даних і складніші сценарії.

Для підготовки системи до експерименту була проведена ретельна робота зі створення необхідних тестових даних. Зокрема, були заповнені довідники продуктів і страв, а також сформована тестова продуктова розкладка на 4 дні, яка базується на одній із реальних розкладок для гірського походу. Вона включає особливості харчування й потреби у продуктах на кожен день, містить детальний розподіл страв і продуктів для чотирьох днів, а саме, сніданки, обіди, вечері та перекуси (рис 3.1). Це дозволило забезпечити реалістичність і відповідність експериментальних даних до реальних умов використання системи.

**Учасники**

| Ім'я  | Опис | Коефіцієнт | Змін. | Видал. |
|-------|------|------------|-------|--------|
| Антон |      | 1          | Змін. | Видал. |
| Ігор  |      | 1          | Змін. | Видал. |
| Ольга |      | 0.9        | Змін. | Видал. |

Додати учасника

|                | Калорії        | Протеїни      | Жири         | Вуглєд.        | Вага        |
|----------------|----------------|---------------|--------------|----------------|-------------|
| <b>Загалом</b> | <b>7067.85</b> | <b>183.26</b> | <b>134.6</b> | <b>1106.24</b> | <b>6189</b> |

20-01-2025

День 1. Заброска у долину Кизил-Суу

Змінити

Видалити

**Обід**

Страви

| Назва                       | Калорії | Протеїни | Жири | Вуглєд. | Вага | Пакунки  |
|-----------------------------|---------|----------|------|---------|------|----------|
| <i>компот з солодощами</i>  |         |          |      |         |      |          |
| Ягоди сублімовані           | 0.9     | 0        | 0    | 0.2     | 15   | чай      |
| Цукор                       | 16      | 0        | 0    | 3.99    | 20   | цукор    |
| Цукерки                     | 3.5     | 0.03     | 0.07 | 0.85    | 10   | солодощі |
| <i>рисовий суп з м'ясом</i> |         |          |      |         |      |          |
| рис                         | 13.4    | 0.27     | 0.04 | 2.91    | 20   | супи     |
| картопля сушена             | 7.05    | 0.14     | 0    | 1.63    | 15   | супи     |
| М'ясо сублімоване           | 42      | 1.2      | 2    | 0       | 20   | тушенка  |
| Морква сушена               | 0.1     | 0.01     | 0    | 0.02    | 2    | супи     |
| Цибуля ріпчаста сушена      | 0.24    | 0.01     | 0    | 0.04    | 3    | супи     |
| Перець солодий сушений      | 0.01    | 0        | 0    | 0       | 1    | супи     |

Рисунок 3.1 - Фрагмент розкладки на 3 учасників.

У рамках підготовки до експерименту в систему було заведено пакунки, які слугують основними одиницями розподілення продуктів між учасниками подорожі. Усі продукти були пов'язані з пакунками. Цей крок дуже важливий через те, що розподілення працює тільки для продуктів пов'язаних з пакунками.

З вищенаведеними даними була проведена частина експериментів з розподіленням 9 пакунків серед 3 і 4 учасників подорожі. Далі було додано додатковий пакунок (рис 3.2).

Назад до розкладки

Пакунки продуктів

| Назва    | Опис | Об'єм | Вага |       |        |
|----------|------|-------|------|-------|--------|
| спеції   |      | 1.2   | 10   | Змін. | Видал. |
| чай      |      | 1.2   | 25   | Змін. | Видал. |
| солодощі |      | 1     | 30   | Змін. | Видал. |
| цукор    |      | 0.9   | 70   | Змін. | Видал. |
| тушенка  |      | 0.9   | 80   | Змін. | Видал. |
| супи     |      | 1.2   | 30   | Змін. | Видал. |
| сухарі   |      | 1.4   | 25   | Змін. | Видал. |
| каши     |      | 0.9   | 25   | Змін. | Видал. |
| сир      |      | 1     | 70   | Змін. | Видал. |
| сухарі2  |      | 1.4   | 25   | Змін. | Видал. |

Додати пакунок

Рисунок 3.2 - Пакунки, заведені в систему для експерименту.

Після додавання нового пакунку довелось трохи перерозподілити продукти між пакунками для завантаження нового пакунку продуктами, без додавання нових продуктів до розкладки. Це було зроблено для збереження ваги розкладки в розрахунку на одну людину і кращого співставлення результатів усіх експериментів (рис 3.3).

Учасники

| Ім'я  | Опис | Коефіцієнт |       |        |
|-------|------|------------|-------|--------|
| Антон |      | 1          | Змін. | Видал. |
| Ігор  |      | 1          | Змін. | Видал. |
| Ольга |      | 0.9        | Змін. | Видал. |
| Марія |      | 0.9        | Змін. | Видал. |

Додати учасника

|         |         |          |        |         |      |
|---------|---------|----------|--------|---------|------|
| Загалом | Калорії | Протеїни | Жири   | Вуглед  | Вага |
|         | 9423.8  | 244.34   | 179.47 | 1474.99 | 8252 |

20-01-2025

Закрити

Видалити

Обід

Страви

| Назва                   | Калорії | Протеїни | Жири | Вуглад. | Вага | Пакунок  |
|-------------------------|---------|----------|------|---------|------|----------|
| компот з солодощами     |         |          |      |         |      |          |
| Ягоди сушімовані        | 0.9     | 0        | 0    | 0.2     | 15   | чай      |
| Цукор                   | 16      | 0        | 0    | 3.99    | 20   | цукор    |
| Цукерки                 | 3.5     | 0.03     | 0.07 | 0.85    | 10   | солодощі |
| рисовий суп з м'ясом    |         |          |      |         |      |          |
| рис                     | 13.4    | 0.27     | 0.04 | 2.91    | 20   | супи     |
| картопля сушена         | 7.05    | 0.14     | 0    | 1.63    | 15   | супи     |
| М'ясо сушімоване        | 42      | 1.2      | 2    | 0       | 20   | тушенка  |
| Морква сушена           | 0.1     | 0.01     | 0    | 0.02    | 2    | супи     |
| Цибуля ріпчаста сушена  | 0.24    | 0.01     | 0    | 0.04    | 3    | супи     |
| Перець: солодий сушений | 0.01    | 0        | 0    | 0       | 1    | супи     |

Рисунок 3.3 - Фрагмент розкладки на 4 учасників.

### 3.2 Мета дослідження і організація експериментів

Метою даного дослідження є визначення прийняттого рівня припустимої похибки в алгоритмі розподілення продуктів, який забезпечує найкращий баланс між швидкістю та точністю розподілення. Основна ідея полягає в тому, щоб знайти значення припустимої похибки, при якому алгоритм забезпечує рівномірний розподіл ваги між учасниками туристичної подорожі, зберігаючи при цьому мінімальну кількість оброблених комбінацій.

Критеріями ефективності обрані:

- точність характеризується значенням середньоквадратичного відхилення від середньої ваги розкладки на одного учасника. Чим менше це відхилення, тим рівномірніше розподілення;
- швидкодія вимірюється кількістю оброблених комбінацій. Чим менше комбінацій необхідно обробити, тим швидше алгоритм знаходить ефективне рішення.

Експерименти проводилися шляхом поступового зменшення припустимої похибки, починаючи з 10% від середньої ваги розкладки на людину. На кожному етапі оцінювалися точність та швидкість роботи алгоритму. Зменшення похибки тривало до тих пір, поки кількість оброблених комбінацій не починала зростати експоненціально, що вказує на суттєве збільшення часу виконання.

Експерименти проводилися на тестових даних, які включали розкладку продуктів для 3–4 учасників із загальною кількістю пакунків 9–10. Це дозволило створити умови, максимально наближені до реальних, але контрольовані для детального аналізу результатів. На кожному етапі змінювалася лише одна змінна — рівень припустимої похибки, тоді як інші параметри, такі як кількість учасників, пакунків і загальна вага, залишалися фіксованими.

Для кожного рівня похибки алгоритм виконував розподіл продуктів і обчислював середньоквадратичне відхилення ваги від середньої. Також підраховувалася кількість комбінацій, які алгоритм опрацьовував для

досягнення ефективного рішення. Це дозволило отримати порівняльні дані, які демонструють, як похибка впливає на точність і швидкодію.

### 3.3 Результати експериментів

Для 9 пакунків і 3 учасників були отримані результати які зазначені у таблиці 3.1. Також було створено графіки точності розподілення в залежності від похибки алгоритму на рисунку 3.4 та кількості оброблених комбінацій в залежності від похибки на рисунку 3.5.

Таблиця 3.1 - Результат експерименту для 9 пакунків і 3 учасників.

| % похибки | похибка алгоритму | відхилення | оброблено комбінацій |
|-----------|-------------------|------------|----------------------|
| 10        | 186.9             | 131.47     | 37                   |
| 5         | 93.43             | 81.36      | 181                  |
| 3         | 56.82             | 64.2       | 5058                 |

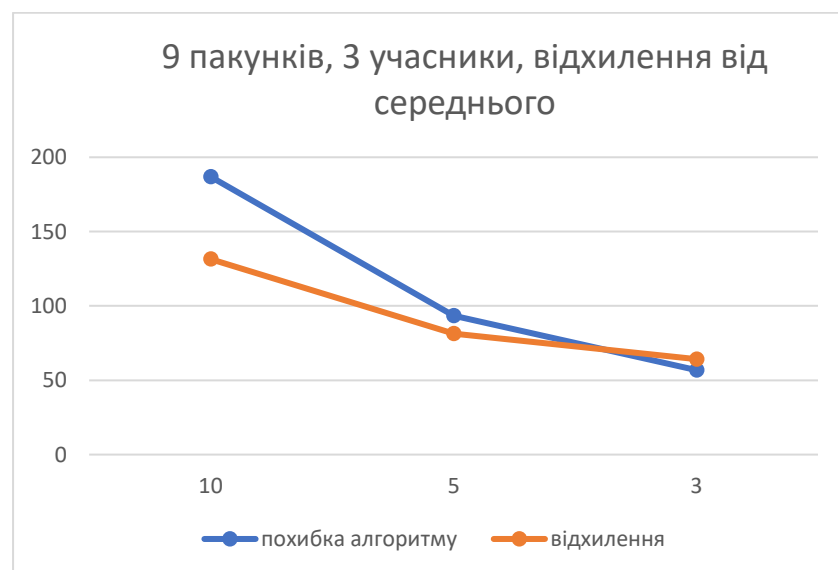


Рисунок 3.4 - Точність розподілення в залежності від похибки алгоритму до таблиці 3.1

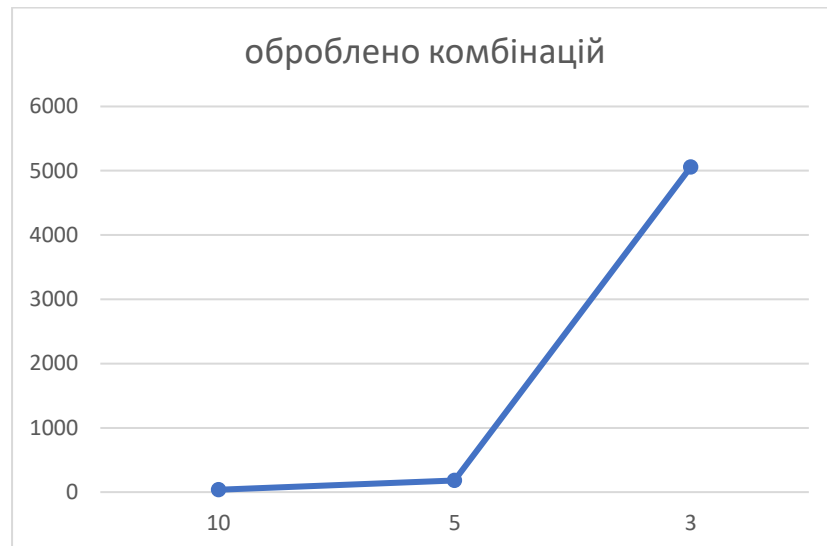


Рисунок 3.5 - Кількість оброблених комбінацій в залежності від похибки алгоритму до таблиці 3.1

Отриманні результати підтверджують очікувану залежність між похибкою алгоритму, середньоквадратичним відхиленням і кількістю оброблених комбінацій. Значення похибки в 10% забезпечує швидкий пошук (37 комбінацій), але точність може бути покращеною (відхилення 131.47). Похибка 5% є компромісним варіантом, оскільки значення відхилення (81.36) стає суттєво меншим, а кількість оброблених комбінацій (181) залишається помірною. Похибка в 3% забезпечує найкращу точність (відхилення 64.2), але ціною значного зростання часу обчислень. Для 10 пакунків і 3 учасників були отримані наступні результати описані в таблиці 3.2 з графіками точності розподілення в залежності від похибки алгоритму на рисунку 3.6 та кількості оброблених комбінацій в залежності від похибки на рисунку 3.7.

Таблиця 3.2 - результат експерименту для 10 пакунків і 3 учасників.

| % похибки | похибка алгоритму | відхилення | оброблено комбінацій |
|-----------|-------------------|------------|----------------------|
| 10        | 186.9             | 168.37     | 77                   |
| 5         | 93.43             | 79.83      | 442                  |
| 3         | 56.82             | 63.11      | 118602               |

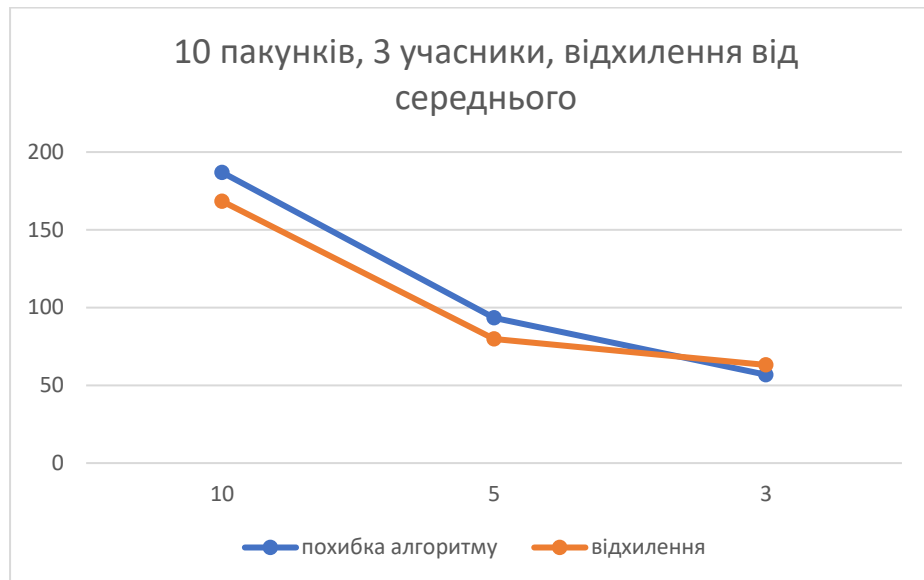


Рисунок 3.6 - точність розподілення в залежності від похибки алгоритму до таблиці 3.2

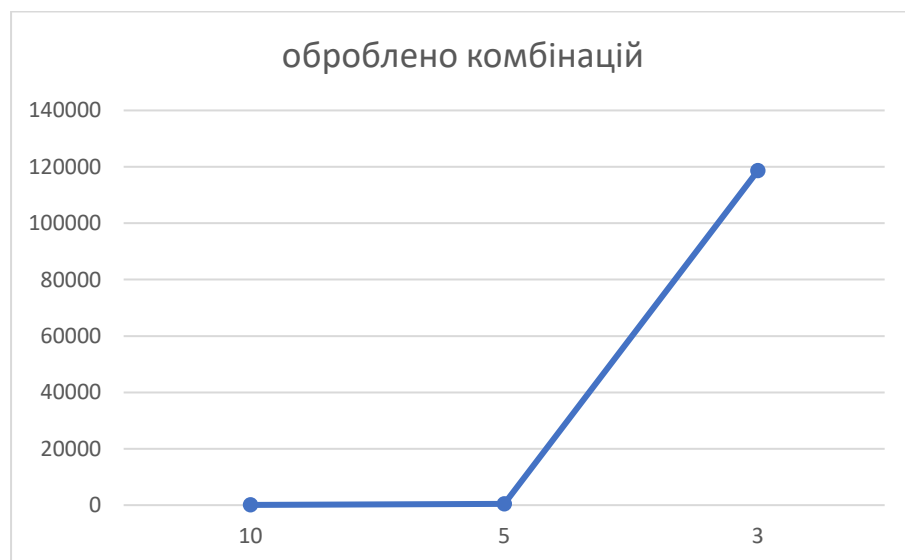


Рисунок 3.7 - кількість оброблених комбінацій в залежності від похибки алгоритму до таблиці 3.2

Отриманні результати підтверджують попередні висновки, а також додають нову інформацію.

У нових даних кількість оброблених комбінацій суттєво зростає:

- при похибці 3% кількість комбінацій зростає з 5058 до 118602, що демонструє значно більшу обчислювальну складність;

- при похибці 5% оброблено 442 комбінації замість 181, що також вказує на збільшення навантаження;
- для похибки 10% кількість комбінацій збільшилася з 37 до 77, але це все ще залишається відносно низьким значенням.

Варто зазначити, що у порівнянні з попередніми результатами, нові дані демонструють суттєве покращення точності. Наприклад, при похибці 5% відхилення зменшилося з 81.36 до 79.83, а при 3% — з 64.2 до 63.11. Це свідчить про те, що алгоритм може досягати більшої точності, але ціною значного збільшення кількості обчислень.

Таким чином, можемо зробити висновок, що збільшення кількості пакунків допомагає досягти кращої точності, але ціною значного збільшення обчислень.

Для 9 пакунків і 4 учасників були отримані наступні результати в таблиці 3.3 з графіками точності розподілення в залежності від похибки алгоритму на рисунку 3.8 та кількості оброблених комбінацій в залежності від похибки на рисунку 3.9.

Таблиця 3.3 - результат експерименту для 9 пакунків і 4 учасників.

| % похибки | похибка алгоритму | відхилення | оброблено комбінацій |
|-----------|-------------------|------------|----------------------|
| 10        | 186.9             | 122.4      | 469                  |
| 7         | 129.12            | 122.4      | 469                  |
| 5         | 93.43             | 103.87     | 124687               |

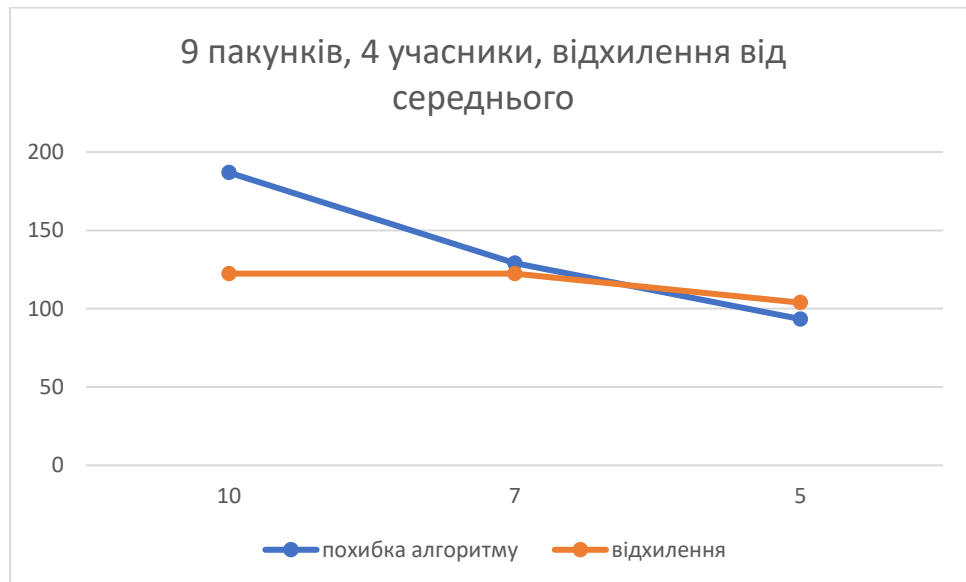


Рисунок 3.8 точність розподілення в залежності від похибки алгоритму для таблиці 3.3

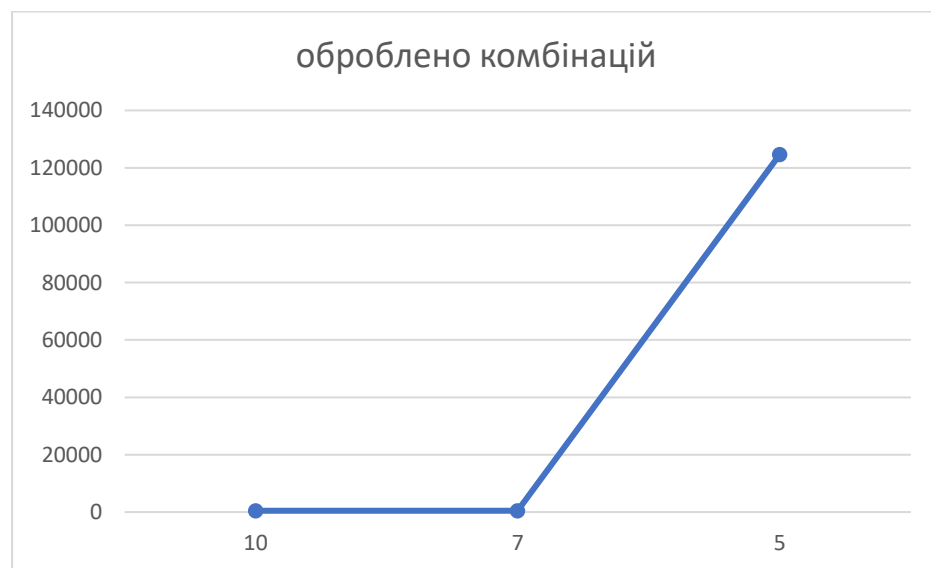


Рисунок 3.9 - Кількість оброблених комбінацій в залежності від похибки алгоритму до таблиці 3.3

Спираючись на отриманні данні можемо зробити наступні висновки:

- у попередньому експерименті з 3 учасниками кількість комбінацій для 10% похибки була значно меншою (77), що свідчить про зростання обчислювальної складності при збільшені кількості учасників. Більша кількість

учасників означає, що для досягнення рівномірності розподілення алгоритм опрацьовує більше варіантів;

– у новому експерименті кількість оброблених комбінацій для похибки 5% становить 124687, що вже є експоненціальним зростанням складності, яке досягалось при зменшенні похибки до 3%;

– у новому експерименті для алгоритм демонструє стабільні результати для похибок 10% і 7%, що може вказувати на нерівномірне розподілення комбінацій під час генерування і наявність простору для покращення.

Для 10 пакунків і 4 учасників були отримані наступні результати в таблиці 3.4 з графіками точності розподілення в залежності від похибки алгоритму на рисунку 3.10 та кількості оброблених комбінацій в залежності від похибки на рисунку 3.11.

Таблиця 3.4 - результат експерименту для 10 пакунків і 4 учасників

| % похибки | похибка алгоритму | відхилення | оброблено комбінацій |
|-----------|-------------------|------------|----------------------|
| 10        | 186.9             | 133.62     | 518                  |
| 7         | 129.12            | 117.67     | 12945                |
| 5         | 93.43             | 108.19     | 158013               |

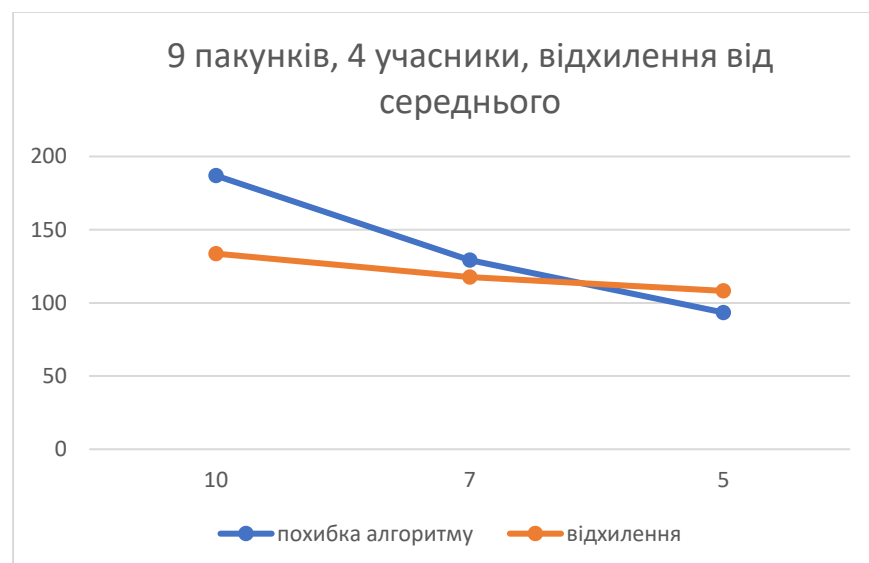


Рисунок 3.10 - точність розподілення в залежності від похибки алгоритму.

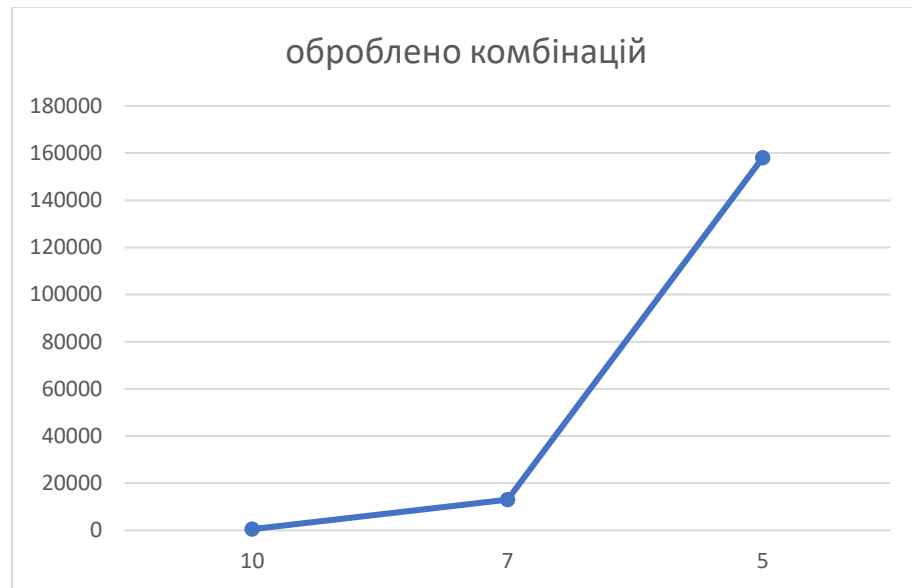


Рисунок 3.11 - кількість оброблених комбінацій в залежності від похибки алгоритму.

Отриманні результати підтверджують продемонстровану в попередніх експериментах залежність між похибкою алгоритму, середньоквадратичним відхиленням і кількістю оброблених комбінацій.

У порівнянні з попередніми експериментами для меншої кількості пакунків (9 шт.) кількість комбінацій для похибки 7% зросла суттєво, 12 945 проти 469 раніше. Це пояснюється більшою кількістю варіантів розподілення, які потрібно опрацювати при збільшенні кількості пакунків.

## ВИСНОВКИ

Проведені чотири експерименти продемонстрували важливість вибору похибки та вплив параметрів задачі (кількість учасників і пакунків) на ефективність алгоритму. Ефективний вибір рівня похибки залежить від пріоритетів: швидкість чи точність.

Підтверджено залежність між похибкою алгоритму, середньоквадратичним відхиленням і кількістю оброблених комбінацій. Зменшення рівня похибки систематично призводить до покращення точності розподілення. Середньоквадратичне відхилення зменшується при переході від похибки 10% до 5%, що підтверджує ефективність алгоритму у забезпеченні більш рівномірного розподілення ваги між учасниками. Однак приріст точності залежить від конкретного сценарію:

- у випадку з 4 учасниками точність суттєво покращується навіть при переході від 10% до 7%, що свідчить про велику кількість варіантів, які враховуються;

- у випадку 3 учасників зміна між 10% і 7% є менш значною, оскільки кількість варіантів обмежена.

Кількість оброблених комбінацій різко зростає зі зменшенням похибки:

- для похибки 10% кількість комбінацій залишається низькою (від 37 до 518 у різних експериментах), що забезпечує швидкий час виконання;

- для похибки 7% кількість комбінацій значно зростає (від кількох сотень до понад 12,000);

- для похибки 5% оброблені комбінації перевищують 100 000, що вимагає значних обчислювальних ресурсів і значно збільшує час виконання.

Збільшення кількості учасників додає складності алгоритму значно більше ніж збільшення кількості пакунків.

Результати отримані при дослідженні свідчать, що збільшення кількості пакетів призводить до відносно невеликого зростання кількості оброблених

комбінацій, але значно покращує точність. Тим часом, додавання додаткового учасника значно збільшує кількість комбінацій для обчислення.

Результати дослідження черговий раз доводять важливість раннього відсікання завідомо неефективних розподілень для досягнення прийнятного рівня ефективності алгоритму.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

- 1     Топорков Олександр, Тетяна Гриньова. Особливості харчування у гірських спортивних походах. 2024.
- 2     Решетило Л. І., Сибірний А. В., Особливості харчування в туристичних походах. Сталий розвиток туризму на засадах партнерства: Освіта, Наука, Практика 2018: 117.
- 3     Шаленко, Віктор Васильович. Про проблеми харчування туристів. Основи спортивного туризму в рекреаційній діяльності, 2018, 133-140.
- 4     Застосунок Calories in food [Електронний ресурс] - URL: <https://play.google.com/store/apps/details?id=com.food.calories> (дата звернення: 22.10.2024).
- 5     The Ultimate Backpacking Calorie & Food Weight Calculator: Plan Your Next Outdoor Camping. [Електронний ресурс] - URL: <https://www.safecastle.com/blogs/safecastle-blog/the-ultimate-backpacking-food-calculator-plan-for-your-outdoor-camping> (дата звернення: 22.10.2024).
- 6     ЯБ Поїла. [Електронний ресурс] - URL: [https://ua.yabpoela.com/product\\_analyzer.html](https://ua.yabpoela.com/product_analyzer.html) (дата звернення: 22.10.2024).
- 7     Richards, Mark, and Neal Ford. Fundamentals of Software Architecture. O'Reilly Media, 2020.
- 8     Bass, Len, Paul Clements, and Rick Kazman. Software Architecture in Practice. 4<sup>th</sup> Edition, O'Reilly, 2021.
- 9     Newman, Sam. Building Microservices. O'Reilly Media, 2015.
- 10    Fowler, Martin, and James Lewis. Microservices: A Definition of This New Architectural Term. ThoughtWorks, 2014.
- 11    Krasner, David Mark Clements. Node.js Design Patterns: Design and implement production-grade Node.js applications using proven patterns and techniques. Packt Publishing, 2020.

- 12 Vincent, Nigel George. Django for Professionals: Production websites with Python & Django. Leanpub, 2020.
- 13 Raja, Mohamed Shazin Sadakath. Hands-On Microservices with Spring Boot and Spring Cloud. Packt Publishing, 2019.
- 14 Esposito, Dino. ASP.NET Core 5 for Beginners: Kick-start your ASP.NET web development journey. Packt Publishing, 2021.
- 15 Olsen, Obie. The Rails 5 Way. Addison-Wesley, 2017.
- 16 Mardan, Azat. React Quickly: Painless web apps with React, JSX, Redux, and GraphQL. Manning Publications, 2017.
- 17 Farhi, Doguhan Uluca. Angular for Enterprise-Ready Web Applications. Packt Publishing, 2018.
- 18 Lin, Anthony Gore. Fullstack Vue: The Complete Guide to Vue.js. Fullstack.io, 2018.
- 19 Joyce, Mark. Svelte and Sapper in Action. Manning Publications, 2021.
- 20 Arslan, Marco L. Napoli. Beginning Flutter: A Hands-On Guide to App Development. Apress, 2019.
- 21 Juan, Dennis Sheppard. Progressive Web Application Development by Example. Packt Publishing, 2018.

## ДОДАТОК А

### Лістинг А.1 – Маршрутизація веб клієнта /router/index.js

```
Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/',
      name: 'HomePage',
      component: HomePage
    },
    {
      path: '/about',
      name: 'AboutPage',
      component: AboutPage
    },
    {
      path: '/directory',
      component: DirectoryPage,
      redirect: {
        name: 'ProductsPage'
      },
      children: [
        {
          path: 'products',
          name: 'ProductsPage',
          component: Products
        },
        {
          path: 'dishes',
          name: 'DishesPage',
          component: Dishes
        },
      ],
    },
  ],
})
```

```

    {
      path: 'dish-edit',
      name: 'EditDishPage',
      component: EditDish,
      props: true
    },
    {
      path: 'dish-template-edit',
      name: 'EditTemplateDishPage',
      component: EditTemplateDish,
      props: true
    },
    {
      path: 'product-categories',
      name: 'ProductCategoriesPage',
      component: ProductCategories
    },
    {
      path: 'dish-categories',
      name: 'DishCategoriesPage',
      component: DishCategories
    },
    {
      path: 'meal-types',
      name: 'MealTypesPage',
      component: MealTypes
    }
  ]
},
{
  path: '/plan/:planId',
  name: 'FoodPlanPage',
  component: FoodPlan
},
{

```

```

    path: '/plan/:planId/packages',
    name: 'FoodPackagesPage',
    component: FoodPackage
  },
  {
    path: '/plan/:planId/day/:dayId',
    name: 'FoodDayPage',
    component: FoodDay
  },
  {
    path: '/plan/:planId/day/:dayId/meal/:mealId',
    name: 'MealPage',
    component: EditMeal
  }
]
}))

```

## Лістинг А.2 – Компонент HikerItem.vue

```

<template>
  <div class="row">
    <div class="col-md-3">
      <input v-validate="'required'" v-model="hiker.name"
name="hikerName"
      v-bind:class="{ validationError:
errors.has('hikerName')}"
      placeholder='Введіть тут..' style="width: 100%"/>
      <p v-if="errors.has('hikerName') > 0"
class="alert">{{errors.first('hikerName')}}</p>
    </div>
    <div class="col-md-5">
      <input v-model="hiker.description" name="hikerDescription"
style="width: 100%"/>
    </div>
    <div class="col-md-2">

```

```

        <input type="number" min="0" step="0.1" v-
model="hiker.weightCoefficient" name="hikerCoefficient"
style="width: 100%"/>
    </div>
    <div class="col-sm-1">
        <b-button variant="outline-success" size="sm" v-
on:click="updateHiker">Змін./b-button>
    </div>
    <div class="col-sm-1">
        <b-button variant="outline-danger" size="sm" v-
on:click="deleteHiker">Видал./b-button>
    </div>
</div>
</template>

<script>
export default {
  name: 'HikerItem',
  props: {
    hiker: {
      type: Object,
      required: true
    }
  },
  methods: {
    updateHiker () {
      this.$validator.validateAll().then((result) => {
        if (result) {
          let member = {
            id: this.hiker.id,
            name: this.hiker.name,
            description: this.hiker.description,
            weightCoefficient: this.hiker.weightCoefficient
          }

```

```

        this.$emit('update', member)
    } else {
        console.log('Couldn\'t update Member due to validation
errors')
    }
})
},

deleteHiker () {
    this.$emit('remove', this.hiker.id)
}
}
</script>

```

### Лістинг А.3 – компонент HikersView.vue

```

<template>
  <div>
    <h3 class="members-header">Учасники</h3>
    <!--hikers \ members-->
    <div v-if="members.length > 0" class="container">
      <!--Header-->
      <div class="row headerRow bg-light">
        <div class="col-md-3 border"><strong>Ім'я</strong></div>
        <div class="col-md-5 border"><strong>Опис</strong></div>
        <div class="col-md-2
border"><strong>Коефіцієнт</strong></div>
      </div>
      <div v-for="hiker in members" :key="'hiker-' + hiker.id">
        <hiker-item v-bind:hiker = "hiker"
          v-on:update="updateMember"
          v-on:remove="removeMember"/>
      </div>
    </div>
    <!--Errors output-->
  </div>

```

```

<div v-if="errorMessage !== null" class="alert">
  <p>{{errorMessage}}</p>
</div>
<!--Add new hiker section-->
<b-button variant="link" size="sm" v-on:click="addMode =
!addMode">Додати учасника</b-button>
<div v-if="addMode !== undefined && addMode" class="container">
  <div class="row">
    <div class="col-md-4"/>
    <div class="col-md-2 border bg-
light"><strong>Ім'я</strong></div>
    <div>
      <input v-validate="'required'" v-model="newHiker"
name="newHiker"
      v-bind:class="{ validationError:
errors.has('newHiker')}"
      placeholder='Введіть тут..' style="width: 100%"/>
      <p v-if="errors.has('newHiker') > 0"
class="alert">{{errors.first('newHiker')}}</p>
    </div>
  </div>
  <div class="row" style="padding-top:10px;padding-bottom:15px">
    <div class="col-md-5"/>
    <div class="col-md-1">
      <b-button variant="outline-success" size="sm" v-
on:click="addNewHiker">Додати</b-button>
    </div>
    <div class="col-md-1">
      <b-button variant="outline-danger" size="sm" v-
on:click="addMode = false">Назад</b-button>
    </div>
  </div>
</div>
</div>
</template>

```

```

<script>
import axios from 'axios'
import HikerItem from 'src/components/plan/HikerItem.vue'

export default {
  name: 'HikersView',
  components: {HikerItem},
  props: {
    hikers: {
      type: Array,
      required: true
    }
  },

  data () {
    return {
      hikersEndpointUrl: '/api/plans/' + this.$route.params.planId +
'/hikers/',
      addMode: false,
      members: this.hikers,
      newHiker: null,
      errorMessage: null
    }
  },

  methods: {
    getErrorMessage (error, defaultMessage) {
      if (error.response !== undefined && error.response.data !==
undefined &&
        (typeof error.response.data === 'string' ||
error.response.data instanceof String)) {
        this.errorMessage = error.response.data
      } else {
        console.log(error)
      }
    }
  }
}

```

```

        this.errorMessage = defaultMessage
    }
},

updateMember (member) {
    axios.put(this.hikersEndpointUrl + member.id, member)
        .then(() => {
            this.errorMessage = null
        })
        .catch(e => {
            this.getErrorMessage(e, 'Failed to update member ' +
JSON.stringify(member))
        })
},

removeMember (hikerId) {
    axios.delete(this.hikersEndpointUrl + hikerId)
        .then(() => {
            this.members = this.members.filter(hiker => hiker.id !==
hikerId)
            this.errorMessage = null
        })
        .catch(e => {
            let member = this.members.find(hiker => hiker.id ===
hikerId)
            this.getErrorMessage(e, 'Failed to delete ' + member.name)
        })
},

addNewHiker () {
    this.$validator.validateAll().then((result) => {
        if (result) {
            let hiker = {
                name: this.newHiker,
                weightCoefficient: 1.0
            }
        }
    })
}

```

```

    }
    axios.post(this.hikersEndpointUrl, hiker)
      .then(response => {
        console.log('New Hiker - ' +
JSON.stringify(response.data))
        this.addMode = false
        this.errorMessage = null
        this.newHiker = ''
        this.members.push(response.data)
      })
      .catch(e => {
        this.getErrorMessage(e, 'Failed to add member ' +
JSON.stringify(hiker))
      })
  } else {
    console.log('Couldn\'t add new member due to validation
errors')
  }
})
}
}
}
</script>

<style scoped>
.members-header{
  text-align: center;
  color: #ff0045;
}
</style>

```

#### Лістинг А.4 – клас HikerDomainService

```

@Service
public class HikerDomainService {

```

```

private final IFoodPlanRepo planRepo;
private final IHikerRepo hikerRepo;

public HikerDomainService(IFoodPlanRepo planRepo, IHikerRepo
hikerRepo) {
    this.planRepo = planRepo;
    this.hikerRepo = hikerRepo;
}

public List<Hiker> getPlanHikers(long planId) {
    return hikerRepo.getPlanHikers(planId);
}

public Optional<Hiker> geHiker(long id) {
    return hikerRepo.getHiker(id);
}

public Hiker addHiker(long planId, Hiker hiker) {
    long id = hikerRepo.addHiker(planId, hiker);
    planRepo.saveLastUpdated(planId, ZonedDateTime.now());
    return hiker.toBuilder()
        .id(id)
        .build();
}

public void updateHiker(long planId, Hiker hiker) {
    if (!hikerRepo.existsHiker(hiker.getId())) {
        throw new NotFoundException("Hiker with id=" +
hiker.getId() + " doesn't exist");
    }
    if(!hikerRepo.updateHiker(hiker)) {
        throw new FoodcalcDomainException("Failed to update
hiker with id=" + hiker.getId());
    }
}

```

```

        planRepo.saveLastUpdated(planId, ZonedDateTime.now());
    }

    public void deleteHiker(long planId, long id) {
        if (!hikerRepo.existsHiker(id)) {
            throw new NotFoundException("Hiker with id=" + id + "
doesn't exist");
        }
        hikerRepo.deleteHiker(id);
        planRepo.saveLastUpdated(planId, ZonedDateTime.now());
    }
}

```

### Лістинг А.5 – клас HikerRepo

```

@Repository
public class HikerRepo extends AbstractRepository<Hiker> implements
IHikerRepo, RowMapper<Hiker> {

    static final String SELECT_PLAN_HIKER_SQL = "select h.id as
hikerId, h.name as name, h.description as description, h.weight_coef
as wCoef " +
        "from hiker h " +
        "where plan = :planId";

    static final String SELECT_HIKER_SQL = "select h.id as hikerId,
h.name as name, h.description as description, h.weight_coef as wCoef
" +
        "from hiker h " +
        "where h.id = :hikerId";

    static final String INSERT_HIKER_SQL = "insert into hiker (name,
description, weight_coef, plan) " +
        "values (:name, :description, :wCoef, :planId)";

```

```

        static final String UPDATE_HIKER_SQL = "update hiker set name =
:name, description = :description, weight_coef = :wCoef " +
            "where id = :hikerId";

        static final String DELETE_HIKER_SQL = "delete from hiker where
id = :hikerId";

        static final String SELECT_HIKER_EXIST_SQL = "select count(*)
from hiker where id = :hikerId";

        @Override
        public List<Hiker> getPlanHikers(long planId) {
            SqlParameterSource parameters = new MapSqlParameterSource()
                .addValue("planId", planId);
            return jdbcTemplate.query(SELECT_PLAN_HIKER_SQL, parameters,
this);
        }

        @Override
        public Optional<Hiker> getHiker(long id) {
            SqlParameterSource parameters = new MapSqlParameterSource()
                .addValue("hikerId", id);
            return getSingleObject(SELECT_HIKER_SQL, parameters, this);
        }

        MapSqlParameterSource getSqlParameterSource(Hiker hiker) {
            return new MapSqlParameterSource()
                .addValue("name", hiker.getName())
                .addValue("description", hiker.getDescription())
                .addValue("wCoef", hiker.getWeightCoefficient());
        }

        @Override
        public long addHiker(long planId, Hiker hiker) {
            KeyHolder holder = new GeneratedKeyHolder();

```

```

        MapSqlParameterSource parameters =
getSqlParameterSource(hiker)
            .addValue("planId", planId);
        jdbcTemplate.update(INSERT_HIKER_SQL, parameters, holder,
new String[] {"id"});
        Number key = holder.getKey();
        return key != null ? key.longValue() : -1L;
    }

    @Override
    public boolean updateHiker(Hiker hiker) {
        MapSqlParameterSource parameters =
getSqlParameterSource(hiker)
            .addValue("hikerId", hiker.getId());
        return jdbcTemplate.update(UPDATE_HIKER_SQL, parameters) >
0;
    }

    @Override
    public void deleteHiker(long id) {
        SqlParameterSource parameters = new
MapSqlParameterSource().addValue("hikerId", id);
        jdbcTemplate.update(DELETE_HIKER_SQL, parameters);
    }

    @Override
    public boolean existsHiker(long id) {
        SqlParameterSource parameters = new
MapSqlParameterSource().addValue("hikerId", id);
        Long count =
jdbcTemplate.queryForObject(SELECT_HIKER_EXIST_SQL, parameters,
Long.class);
        return count != null && count > 0;
    }

```

```

@Override
public Hiker mapRow(ResultSet resultSet, int rowNum) throws
SQLException {
    return Hiker.builder()
        .id(resultSet.getLong("hikerId"))
        .name(resultSet.getString("name"))
        .description(resultSet.getString("description"))
        .weightCoefficient(resultSet.getFloat("wCoef"))
        .build();
}
}

```

### Лістинг А.6 – клас HikerEndpoint

```

@RestController
@RequestMapping("${spring.data.rest.basePath}/plans/{planId}/hikers"
)
public class HikerEndpoint extends AbstractEndpoint {

    private static final Logger LOG =
LoggerFactory.getLogger(HikerEndpoint.class);

    private final HikerService hikerService;

    public HikerEndpoint(HikerService hikerService) {
        this.hikerService = hikerService;
    }

    @GetMapping(path = "{id}", produces = APPLICATION_JSON_VALUE)
    public HikerInfo getHiker(@PathVariable("planId") long planId,
        @PathVariable("id") long id) {
        LOG.debug("Getting hiker plan id = {} id = {}", planId, id);
        return hikerService.getHiker(id);
    }

    @DeleteMapping("{id}")

```

```

@ResponseStatus(HttpStatus.NO_CONTENT)
public void deleteHiker(@PathVariable("planId") long planId,
                        @PathVariable("id") long id) {
    LOG.debug("Removing hiker plan id = {} id - {}", planId,
id);
    hikerService.deleteHiker(planId, id);
}

@PostMapping(consumes = APPLICATION_JSON_VALUE, produces =
APPLICATION_JSON_VALUE)
public HikerInfo addHiker(@PathVariable("planId") long planId,
                          @RequestBody @Valid HikerInfo hiker) {
    LOG.debug("Adding new hiker id = {}", planId);
    return hikerService.addHiker(planId, hiker);
}

@PutMapping(path = "{id}", consumes = APPLICATION_JSON_VALUE)
@ResponseStatus(HttpStatus.NO_CONTENT)
public void updateHiker(@PathVariable("planId") long planId,
                        @PathVariable("id") long id,
                        @RequestBody @Valid HikerInfo hiker) {
    verifyEntityId(id, hiker);
    LOG.debug("Updating hiker plan id = {} id - {}", planId,
id);
    hikerService.updateHiker(planId, hiker);
}
}

```

### Лістинг А.7 – клас FoodDistributionEngine

```

@Service
public class FoodDistributionEngine {

    private final FoodPackageDomainService foodPackageDomainService;
    private final HikerDomainService hikerDomainService;
    private final PlanDayDomainService dayDomainService;

```

```

    public FoodDistributionEngine(FoodPackageDomainService
foodPackageDomainService, HikerDomainService hikerDomainService,
PlanDayDomainService dayDomainService) {
        this.foodPackageDomainService = foodPackageDomainService;
        this.hikerDomainService = hikerDomainService;
        this.dayDomainService = dayDomainService;
    }

    public List<PackageWithProducts>
getPackagesWithProductsForPlan(long planId, int members) {
        var packages =
foodPackageDomainService.getPackagesWithProductsForPlan(planId);
        return packages.values().stream()
            .sorted(Collections.reverseOrder(
                Comparator.comparingDouble(pack ->
pack.getEstimatedWeight(members))))
            .collect(Collectors.toList());
    }

    public FoodDistribution findBestDistribution(FoodPlan plan,
List<PackageWithProducts> packagesWithProductsForPlan) {
        List<Hiker> hikers = plan.getMembers().stream()

.sorted(Comparator.comparingDouble(Hiker::getWeightCoefficient).reve
rsed())

        .collect(Collectors.toList());
        List<PlanDay> planDays = plan.getDays().stream()
            .map(this::copyDayNoProducts)
            .sorted(Comparator.comparing(PlanDay::getDate))
            .collect(Collectors.toList());
        return findBestDistribution(hikers, planDays,
packagesWithProductsForPlan);
    }

```

```

private PlanDay copyDayNoProducts(PlanDay day) {
    return PlanDay.builder()
        .dayId(day.getDayId())
        .date(day.getDate())
        .build();
}

public FoodDistribution findBestDistribution(List<Hiker> hikers,
List<PlanDay> planDays, List<PackageWithProducts>
packagesWithProductsForPlan) {

    if (packagesWithProductsForPlan.isEmpty()) {
        throw new FoodcalcException("Couldn't do food
distribution without packages");
    }

    FoodDistributionsStorage storage = new
FoodDistributionsStorage(
        hikers, planDays, packagesWithProductsForPlan);

    //here we've got set of food distributions for last day
    PlanDay lastDay = planDays.get(0);
    Set<Long> lastDayIds =
Collections.singleton(lastDay.getDayId());
    FoodDistribution emptyHikersDistribution =
FoodDistribution.builder()
        .day(lastDay)
        .allDays(lastDayIds)
        .hikerPackages(hikers.stream()
            .map(hiker -> HikerWithPackages.builder()
                .hiker(hiker)
                .build())
            .collect(Collectors.toList()))
        .build();
    storage.buildInitialDistributionSetForFirstDay(emptyHikersDistributi
on, lastDayIds);

```

```

        //we continue building distributions for pre last day and so
        forth and going to first day searching for best distribution
        var best = storage.recursiveSearch(0, lastDayIds);
        return best.orElse(storage.getBest());
    }
}

```

### Лістинг А.8 – клас FoodDistributionsStorage

```

public class FoodDistributionsStorage {
    private final List<Hiker> allHikers;
    private final List<PlanDay> allDays;
    private final List<PackageWithProducts> allPackages;
    private final Map<Long, List<PackageWithProducts>>
packagesByDay;
    private final List<SameDayFoodDistributions> allDistributions;
    private final double eps;

    public FoodDistributionsStorage(List<Hiker> allHikers,
List<PlanDay> allDays, List<PackageWithProducts> allPackages) {
        this.allHikers = allHikers;
        this.allDays = allDays;
        this.allPackages = allPackages;
        this.allDistributions = new ArrayList<>(allDays.size());
        this.packagesByDay = new HashMap<>(allDays.size());
        //initialize packages and distributions per day storages
        Set<Long> days = new HashSet<>();
        for (PlanDay theDay : this.allDays) {
            days.add(theDay.getDayId());
            double weight =
calculatePackagesEstimatedWeightForDays(days);
            allDistributions.add(SameDayFoodDistributions.builder()
                .currentDay(theDay)
                .allDays(days)
                .dayDistributions(new HashSet<>())

```

```

        .currentWeight(weight)
        .minDeviation(weight / 4) //threshold for
obviously NOT effective distributions
        .build());
    packagesByDay.put(theDay.getDayId(), new ArrayList<>());
}
//put packages into days \ packages map
this.allPackages.forEach(pack ->
pack.getDayProducts().keySet()
    .forEach(k -> packagesByDay.get(k).add(pack)));
//threshold of weight deviation in percentage of mean weight
(10% for now)
    eps = allDistributions.get(allDistributions.size() -
1).getCurrentWeight() / allHikers.size() * 0.1;
}

private double calculatePackagesEstimatedWeightForDays(Set<Long>
days) {
    return allPackages.stream()
        .mapToDouble(pack -> pack.getEstimatedWeight(days,
allHikers.size()))
        .sum();
}

public List<PackageWithProducts> getPackagesForDays(Set<Long>
days) {
    return days.stream()
        .map(packagesByDay::get)
        .flatMap(List::stream)
        .distinct()
        .collect(Collectors.toList());
}

public List<PackageWithProducts> getNewPackagesForDays(Set<Long>
days, long newDay) {

```

```

        Set<Long> previousDays = new HashSet<>(days);
        previousDays.remove(newDay);
        var previousPackageIds = previousDays.stream()
            .map(packagesByDay::get)
            .flatMap(List::stream)
            .map(pack -> pack.getFoodPackage().getId())
            .collect(Collectors.toSet());
        return packagesByDay.get(newDay).stream()
            .filter(pack ->
!previousPackageIds.contains(pack.getFoodPackage().getId()))
            .collect(Collectors.toList());
    }

    //deep copy
    public FoodDistribution copyDistribution(FoodDistribution
distribution) {
        List<HikerWithPackages> newHikerPackages = new
ArrayList<>(distribution.getHikerPackages().size());
        for (HikerWithPackages hiker :
distribution.getHikerPackages()) {
            newHikerPackages.add(copyHikerWithPackages(hiker));
        }
        return FoodDistribution.builder()
            .day(distribution.getDay())
            .allDays(new HashSet<>(distribution.getAllDays()))
            .hikerPackages(newHikerPackages)
            .build();
    }

    public HikerWithPackages copyHikerWithPackages(HikerWithPackages
hikerPackages) {
        HashSet<PackageWithProducts> packagesCopy;
        if (hikerPackages.getPackages() == null) {
            packagesCopy = new HashSet<>();
        } else {

```

```

        packagesCopy = new
HashSet<>(hikerPackages.getPackages());
    }
    return hikerPackages.toBuilder()
        .packages(packagesCopy)
        .build();
}

    public FoodDistribution
copyDistributionWithNewPackage(FoodDistribution distribution, long
hikerId, PackageWithProducts productsPackage) {
    FoodDistribution copy = copyDistribution(distribution);
    for (HikerWithPackages hikerPackages :
copy.getHikerPackages()) {
        if (hikerId == hikerPackages.getHiker().getId()) {
            hikerPackages.getPackages().add(productsPackage);
            break;
        }
    }
    return copy;
}

    public Set<FoodDistribution>
generateDistributionsAddingPackage(FoodDistribution
initialDistribution, List<PackageWithProducts> productsPackage) {
    Set<FoodDistribution> generatedDistributions = new
HashSet<>();
    PackageWithProducts firstPackage = productsPackage.get(0);
    //next set of productPackage distributed across all hikers
    Set<FoodDistribution> nextDistributions = new HashSet<>();
    for (HikerWithPackages hikerPackages :
initialDistribution.getHikerPackages()) {
        FoodDistribution newDistribution =
copyDistributionWithNewPackage(initialDistribution,
hikerPackages.getHiker().getId(), firstPackage);

```

```

        nextDistributions.add(newDistribution);
    }
    if (productsPackage.size() > 1) {
        ArrayList<PackageWithProducts> remainingProducts = new
ArrayList<>(productsPackage);
        remainingProducts.remove(0);
        //continue recursion for remaining packages
        nextDistributions.forEach(
            newDistribution -> {
                Set<FoodDistribution>
nextPackageDistributions =

generateDistributionsAddingPackage(newDistribution,
remainingProducts);

                                //collecting all results in one set

generatedDistributions.addAll(nextPackageDistributions);
                                });
    } else {
        //complete generation
        generatedDistributions.addAll(nextDistributions);
    }
    return generatedDistributions;
}

public void
buildInitialDistributionSetForFirstDay(FoodDistribution
emptyHikersDistribution, Set<Long> days) {
    Set<FoodDistribution> lastDayDistributions =
generateDistributionsAddingPackage(emptyHikersDistribution,
getPackagesForDays(days));
    SameDayFoodDistributions lastDayDistributionsContainer =
allDistributions.get(0);
    lastDayDistributionsContainer.getDayDistributions()
        .addAll(lastDayDistributions.stream())

```

```

        .filter(dist -> dist.deviation() <
lastDayDistributionsContainer.getMinDeviation())
        .collect(Collectors.toSet()));
    }

    Optional<FoodDistribution> recursiveSearch(int ndx, Set<Long>
days) {
        SameDayFoodDistributions container =
allDistributions.get(ndx);
        if (ndx < allDays.size() - 1) {
            //go through unprocessed distribution of the current day
            var currentDistribution =
getFirstUnprocessed(container);
            while (currentDistribution.isPresent()) {
                //build distributions for next day
                Set<Long> nextDays = new HashSet<>(days);
                SameDayFoodDistributions nextDayContainer =
allDistributions.get(ndx + 1);
                long nextDayId = allDays.get(ndx + 1).getDayId();
                nextDays.add(nextDayId);
                FoodDistribution nextDayDistribution =
copyDistribution(currentDistribution.get()).toBuilder()
                    .day(allDays.get(ndx + 1))
                    .allDays(nextDays)
                    .build();
                Set<FoodDistribution> filteredDistributions =
Collections.emptySet();
                List<PackageWithProducts> newPackages =
getNewPackagesForDays(nextDays, nextDayId);
                if (!newPackages.isEmpty()) {
                    Set<FoodDistribution> nextDayDistributions =
generateDistributionsAddingPackage(nextDayDistribution,
newPackages);
                    filteredDistributions =
nextDayDistributions.stream()

```

```

        .filter(dist -> dist.deviation() <
nextDayContainer.getMinDeviation())
        .collect(Collectors.toSet());
    }
    //and continue recursion
    if (filteredDistributions.isEmpty()) {
        //no food packages were added or for some reason
all new distributions were filtered out
        //so we proceed forward with nextDistribution
        filteredDistributions =
Collections.singleton(nextDayDistribution);
    }

nextDayContainer.getDayDistributions().addAll(filteredDistributions)
;

    currentDistribution.get().setProcessed(true);
    //go recursively through all currentDistribution for
next day

    var best = recursiveSearch(ndx + 1, nextDays);
    if (best.isPresent()) {
        return best;
    }
    currentDistribution =
getFirstUnprocessed(container);
}
    return Optional.empty();
} else {
    return finalSearch(container);
}
}

Optional<FoodDistribution> finalSearch(SameDayFoodDistributions
container) {
    //comparing with best distribution and marking as processed
    container.getDayDistributions().stream()

```

```

        .filter(d -> !d.isProcessed())
        .forEach(distribution -> compareWithBest(container,
distribution));
        return Optional.ofNullable(container.getBest())
            .filter(best -> best.deviation() < eps);
    }

    void compareWithBest(SameDayFoodDistributions container,
FoodDistribution distribution) {
        double deviation = distribution.deviation();
        if (deviation < container.getMinDeviation()) {
            container.setBest(distribution);
            container.setMinDeviation(deviation);
        }
        distribution.setProcessed(true);
    }

    Optional<FoodDistribution>
getFirstUnprocessed(SameDayFoodDistributions container) {
        return container.getDayDistributions().stream()
            .filter(d -> !d.isProcessed())
            .findFirst();
    }

    public FoodDistribution getBest() {
        return
Optional.ofNullable(allDistributions.get(allDistributions.size() -
1).getBest())
            .orElseThrow();
    }
}

```