МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет Центр післядипломної освіти
Кафедра Програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

другий (магістерський)

(рівень вищої освіти)

Дослідження методів формування туристичних подорожей

Виконала:
Студентка 2 курсу групи ІПЗздм-20-1
Криворучко М. А.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми Освітньо-наукова

Керівник доц. Вечур О.В.

(посада, прізвище, ініціали)

Допускається до захисту

Зав. Кафедри _____ проф. Дудар З.В.

2022 р.

Харківський національний університет радіоелектроніки

Факультет Центр післядипломної освіти

Кафедра Програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми Освітньо-наукова програма

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«___»_____202_ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Студентки Криворучко Марії Анатоліївни

(прізвище, ім'я, по батькові)

1. Тема роботи: «Дослідження методів формування туристичних подорожей» затверджена наказом університету від «24» березня 2022 р. № 412Ст

2. Термін подання студентом роботи до екзаменаційної комісії «10» травня 2022 р.

3. Вихідні дані до роботи: Технічне завдання, календарний план, методичні вказівки

4. Перелік питань, що потрібно опрацювати в роботі: мета роботи, аналіз предметної галузі, методів і постановка задачі, створення алгоритму, проектування системи та опис програмної реалізації системи.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	31.03.2022	*виконано*
2	Створення алгоритму	07.04.2022	*виконано*
3	Проектування системи	14.04.2022	*виконано*
4	Реалізація системи	26.04.2022	*виконано*
5	Підготовка пояснювальної записки, презентації та доповіді	02.05.2022	*виконано*
6	Нормоконтроль, рецензування	10.05.2022	*виконано*
7	Занесення матеріалів в електронний архів	12.05.2022	*виконано*
8	Попередній захист	16.05.2022	*виконано*
9	Допуск до захисту у зав. кафедри	17.05.2022	*виконано*

Дата видачі завдання 24 березня 2022 р.

Студент _____

(підпис)

Керівник роботи _____ доц. Вечур О.В.

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка 97 с., 15 рис., 12 табл., 19 джерел.

БАГАТОКРИТЕРІАЛЬНА ОПТИМІЗАЦІЯ, ЗАДАЧА КОММІВОЯЖЕРА, ЛІНІЙНА АДДИТИВНА ЗГОРТКА, ПЛАНУВАННЯ, ПОДОРОЖІ, ПРОГРАМНА РЕАЛІЗАЦІЯ, ANDROID, KOTLIN.

Об’єкт дослідження – планування подорожей.

Мета роботи – проаналізувати існуючі сервіси для планування подорожей, запропонувати власний алгоритм пропонування подорожей по містам.

Були використані такі методи – лінійна адитивна згортка з нормалізацією та ваговими критеріями, задача комівояжера.

У результаті роботи були отримані наступні результати – вказані вище методи було використано на деяких етапах алгоритму планування подорожі. Також було проведено експеримент виконуючи кроки алгоритму. А також було спроектовано і реалізовано програмну систему.

ANDROID, KOTLIN, LINEAR ADDITIVE CONVOLUTION, MULTICRITERIA OPTIMIZATION, PLANNING, SOFTWARE IMPLEMENTATION, TRAVEL, TRAVELLING SALESMAN PROBLEM.

The object of research is travel planning.

The purpose of the work is to analyze the existing services for travel planning, to offer own algorithm for choosing trips with multiple cities.

The following methods were used – linear additive convolution with normalization and weighting criteria, the task of the salesman.

As a result of the work, the following results were obtained – the above methods were used at some stages of the travel planning algorithm. An experiment was also performed following the steps of the algorithm. And a system was designed and implemented.

Я, Криворучко Марія Анатоліївна, студент гр. ІПЗздм-20-1, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів формування туристичних подорожей», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлена з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

На сьогодні люди майже не обмежені в подорожах. Тепер більш значним фактором є бажання побачити нові місця, а не технічна можливість це зробити. Ще кілька десятиліть тому подорожувати було не так просто через високі ціни на дорогу та обмаль можливостей отримання візи та квитків на транспорт.

У наш час вирішують не «чи подорожувати?», а скоріш «куди, коли та як?» На ці питання намагаються відповісти численні сервіси, такі як «Trip Advisor», «Booking», сайти туристичних агентств, навіть сервіси Google. Однак, як буде продемонстровано далі, всі ці системи не є досконалими та універсальними, вони мають деякі незручності та обмеження.

Найчастіше турист вирішує куди поїхати, виходячи з попереднього досвіду або його відсутності, культурних інтересів, особистих рис, видів бажаних подорожей (активний, пасивний, змішаний) і, звичайно, бюджету. Відображати список можливих подорожей неважко, якщо вони просто фільтруються за цими атрибутами. Бачимо, що це реалізовано досить гарно у вищезгаданих сервісах. Однак прийняти рішення про місце призначення – це лише половина проблеми, з якою зустрічається користувач. Окрім цього є складний і довгий процес детального планування та підготовки. Крім того, існує декілька типів туристів, серед яких є ті, що полюбляють залишатися на одному місці та досконало його вивчати та ті, що люблять постійно мандрувати країною/містом у пошуках нових вражень, не сидять на місці. Такі незначні переміщення по місцю призначення нечасто враховуються в існуючих службах і тому припадають на власний розсуд користувача. Планування такої детальної логістики подорожі в цілому може бути дуже складним процесом. Мета цієї роботи – зосередитися на аналізі та розробці автоматичних методів оптимізації планування подорожі, а отже, спрощення процесу підготовки перед поїздкою.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі та аналогів

Ким би людина не працювала – їй необхідно використовувати свої відпустки та вихідні задля балансу між роботою та особистим життям [1]. Відпустка найчастіше асоціюється з подорожами, з метою відпочинку та туризму. Люди часто беруть відпустку під час певних святкових заходів, або упродовж певних фестивалів чи святкувань, канікул, які часто проводять з друзями та родиною.

Концепція відпустки – нещодавній винахід, який склався протягом останніх двох століть. Історично ідея подорожей для відпочинку була розкішшю, яку могли дозволити собі лише заможні люди.

У наші часи є багато сайтів, пов’язаних з подорожами та їх плануванням. Коли користувачеві потрібно спланувати відпустку, у першу чергу, вводиться місце призначення. Наприклад, сервіс під назвою «RomeToRio» – допомагає планувати загальні пасажирські маршрути, в «Inspirock» – користувач може встановлювати пункти призначення, а сервіс пропонує повний список доступних громадських заходів у містах, регіонах.

Якщо ж користувач не має бажаного місця призначення, тобто не знає куди йому поїхати, то у нього є три варіанти.

По-перше, це звернення за допомогою до експерта. Існують сайти планування, де можна ввести свої ідеї та побажання, бюджет, потреби, інші параметри, після чого користувач отримує зворотній зв'язок з експертом для подальшого планування подорожі.

Такий функціонал можна зустріти і на сайтах тур агентств, а також прикладом може стати сервіс «Mr Hudson Explores», який зображено на рисунку 1.1. Даний сервіс надає дві опції щодо планування подорожі, але обидві основані на допомозі експерта.

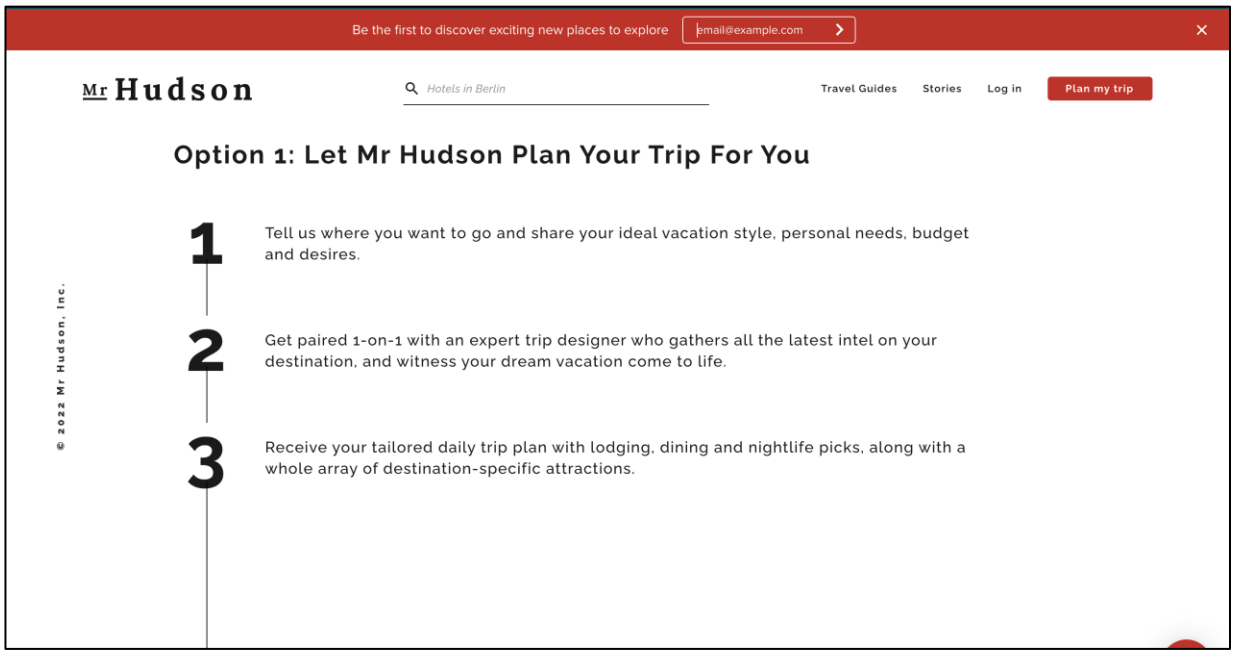

Рисунок 1.1 – Інтерфейс сервісу «Mr Hudson Explores»

По-друге, це використання пошуку авіарейсів із кількома фільтрами, які є досить загальними. Існують такі сайти з можливістю пошуку, у яких також є функціонал застосування деяких загальних фільтрів. Таким чином, загальні фільтри забезпечують неточні результати, які насправді не націлені на конкретного користувача.

Наприклад, деякі місця зовсім не відображено, адже вони належать до ширшої категорії за якимось параметром, хоча за іншими все дуже підходить користувачу.

Серед таких систем є система «Great Escape», яка є агрегатором розкладів рейсів і пошуковою системою з деякими фільтрами. Також прикладом є сервіс під назвою «Tripmydream», де також можна застосувати декілька фільтрів для отримання списку можливих подорожей.

Іще один сервіс називається «Google Trips», де основними функціями є карти, можливість фільтрації за типом транспорту та загальними інтересами, наприклад, популярні заходи, активні заходи, пляжі, музеї, історія, катання на лижах тощо (див. рис. 1.2).

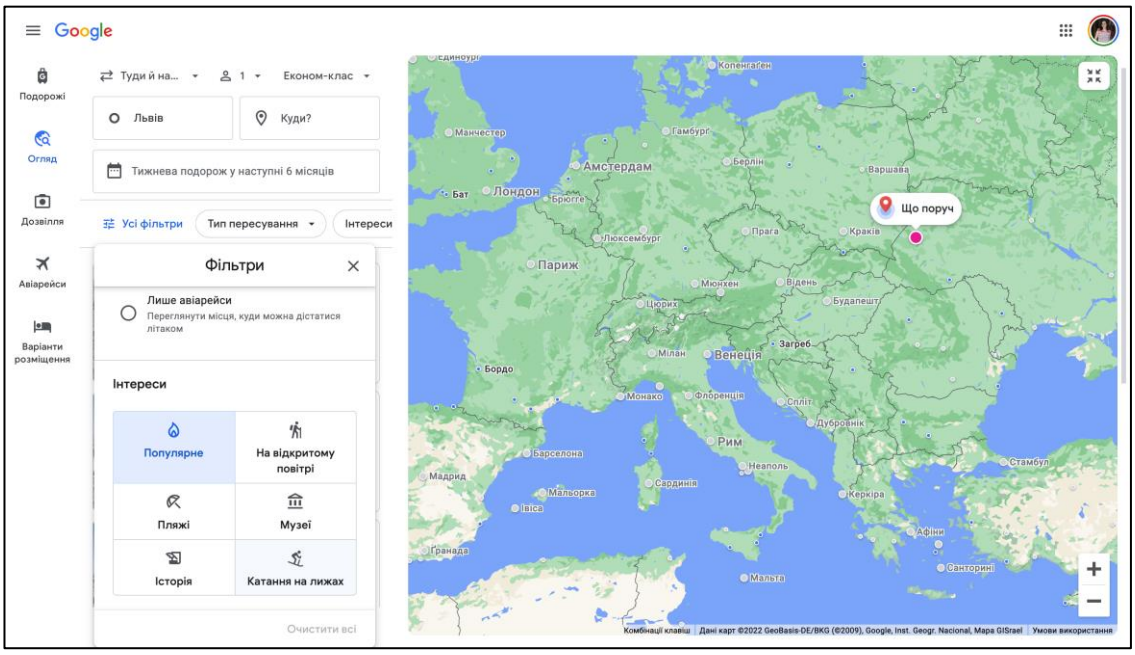

Рисунок 1.2 – Інтерфейс сервісу «Google Trips»

По-третє, це випадкові пропозиції. Іноді це працює, але точно не для всіх і точно не кожного разу. Мова йде про сайти, які випадковим чином пропонують поїздки. Типовим прикладом є сайт під назвою «Eightydays.me», який є сервісом пошуку авіатурів для вибраного регіону. На рисунку 1.3 можемо бачити запропоновану подорож, яка не має обґрунтувань, а також запропоновані Афіни дуже далеко від інших запропонованих місць.

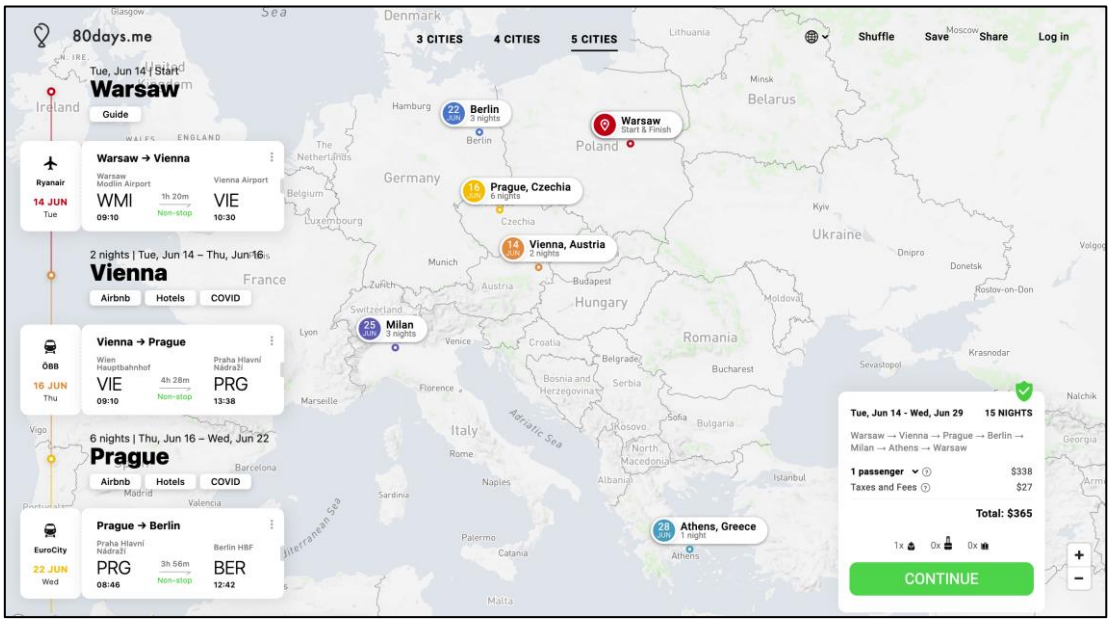

Рисунок 1.3 – Інтерфейс сервісу «Eightydays.me»

Отже, сервіс «Eightydays.me» враховує тільки ціни на квитки, та часто пропонує нелогічні маршрути, чого можна було б уникнути, якби відстані враховувались під час фільтрації.

В таблиці 1.1 зібрано основні сервіси аналоги та їх характеристику.

Таблиця 1.1 – Характеристика розглянутих систем

Сервіси	Можливість спланувати подорож з декількома містами	Врахування побажань користувача	Можливість отримати результат швидко без спілкування з експертами	Вартість
«Mr Hudson Explores»	+	+	-	49$ за день спланованої подорожі
«Google Trips»	-	+-	+	безкоштовно
«Eightydays.me»	+	-	+	~10% від вартості квитків

З таблиці 1.1 можемо бачити, що кожна система має свої недоліки: «Mr Hudson Explores» – висока вартість та необхідність спілкування з експертом, «Google Trips» – неможливість спланувати подорож декількома містами та середній рівень якості врахування побажань, «Eightydays.me» – немає врахування побажань та середня вартість послуги.

У такому разі можна стверджувати, що для користувачів, які не знають куди їхати та не хочуть працювати з експертами, яким важливе врахування побажань та невисока вартість послуги – необхідна вдосконалена система, яка допоможе цьому конкретному сегменту користувачів знайти та розпланувати подорож по різним містам.

1.2 Аналіз існуючих методів

Розглянемо методи, які використовуються для оцінки різних опцій, створення комбінацій та будування маршрутів.

1.2.1 Методи багатокритеріальної оптимізації

При роботі з багатокритеріальною оптимізацією необхідно розуміти наступні терміни: цільова функція та критерій оптимізації. Цільова функція – це функція для якої найменше чи найбільше значення знаходять з врахуванням наявних обмежень. Критерій оптимізації – це властивість об’єкта.

Розглянемо види згорток як методи вирішення задачі багатокритеріальної оптимізації.

Лінійна адитивна згортка дозволяє обчислити ефективність певної стратегії, тобто скільки разів певна стратегія була оптимальною, не враховуючи кількісні показники значень критеріїв. Адитивна згортка особливо корисна, коли зниження оцінки за одним критерієм компенсується збільшенням оцінки за іншим критерієм або за кількома критеріями. Навіть якщо за деякими критеріями оцінка дорівнює нулю, але всі інші критерії мають кращі показники, загальний бал може бути задовільним. Цільова функція при лінійній адитивній згортці визначається наступною формулою:

$$F = \sum_{j=1}^{n} F_j, \tag{1}$$

де $F_j = \begin{cases} 1, & \text{якщо стратегія краща} \\ 0, & \text{якщо гірша} \end{cases}$

Лінійна адитивна згортка з нормуючими множниками дозволяє працювати з кількісними критеріями, які мають різні одиниці [2], як у випадку з плануванням подорожей. Цільова функція виражена наступною формулою:

$$F = \sum_{j=1}^{n} \alpha_j a_{ij}, \tag{2}$$

де $\alpha_j = \frac{1}{\sum_{i=1}^{m} a_{ij}}$ – нормуючі множники,

a_{ij} – значення критеріїв.

Лінійна адитивна згортка з ваговими коефіцієнтами – кожен критерій множиться на свій ваговий коефіцієнт, а потім усі зважені критерії підсумовуються і утворюють зважену цільову функцію, значення якої інтерпретується як «коефіцієнт якості» отриманого рішення (див. формулу 3).

Отримана скаляризована функція максимізується в допустимому діапазоні обмежень [3].

$$F = \sum_{j=1}^{n} \alpha_j \beta_j a_{ij} \tag{3}$$

де $a_j = \frac{1}{\sum_{i=1}^{m} a_{ij}}$ – нормуючі множники,

β_j – вагові коефіцієнти, сума яких дорівнює 1,

a_{ij} – значення критеріїв.

Мультиплікативна згортка заснована на постулаті: «низька оцінка принаймні за одним критерієм призводить до загального низького значення». Отже, можна стверджувати, що мультиплікативна згортка заснована на принципі компенсації відносних змін окремих критеріїв. Формула цільової функції наступна:

$$F = \prod_{j=1}^{n} \alpha_j a_{ij} \quad (4)$$

де $a_j = \frac{1}{\sum_{i=1}^{m} a_{ij}}$ – нормуючі множники,

a_{ij} – значення критеріїв.

Максимінна згортка є найпростішим способом побудови узагальненого критерію, заснованого на застосуванні принципу максиміну. Кожен із критеріїв має власний вимір, і зазвичай вони не збігаються. Тому спочатку необхідно стандартизувати всі наявні виміри в одну систему відліку. Недоліком максимінної згортки є те, що вона враховує лише ті критерії, які дають найменші значення, всі інші критерії ігноруються. Через це така згортка використовується не дуже часто, частіше використовуються лінійні та мультиплікаційні згортки. Але такий підхід завжди дає результат, який гарантовано є мінімальним, нижче якого значення отримані не будуть [4], що також може бути корисним.

1.2.2 Опис методів комбінаторики, які можуть бути застосовані при побудові маршруту

Розглянемо різні методи комбінаторики для створення комбінацій.

«Упорядковані підмножини з k елементів $(b_1, b_2, \dots, b_k)$ множини M називаються перестановками k елементів. Можна сказати, що k-перестановки (b1, b2, …, bk) – це розміщення у визначеному порядку k елементів з множини M» [5]:

$$P_n = n! \quad (5)$$

«k-перестановки множини з n елементів називаються розміщеннями з n по k елементів» [5]:

$$A_n^k = \frac{n!}{(n-k)!} \quad (6)$$

Перестановки з повторенням – це розміщення з повтореннями, які відрізняються тільки порядком елементів у них:

$$\bar{P}(n_1, n_2, \dots, n_k) = \frac{n!}{n_1!\, n_2! \dots n_k!} \quad (7)$$

«Розміщенням з повторенням з n елементів по k називається вибірка з повторенням довжини k з n елементів» [5]:

$$\overline{A_n^k} = n^k \quad (8)$$

«Сполучення без повторень з n елементів по k називається будь-яке k-розміщення з цих елементів, в якому враховується лише склад елементів і не враховується їх порядок» [5]:

$$C_n^k = \frac{n!}{(n-k)!\, k!} \quad (9)$$

Сполученнями з повтореннями із n елементів по k називається сполука, що містить k елементів взятих з даних n елементів серед яких є однакові:

$$\overline{C_n^k} = \frac{(n+k-1)}{(n-1)!\ k!} \quad (10)$$

Для множення отриманих результатів можна використовувати декартів добуток множин. Декартовим добутком множин $X_1 \times X_2 \times \dots \times X_n$ називається множина всіх можливих упорядкованих наборів з n елементів в яких перший елемент належить множині X1, другий – множині X2 і так далі.

1.2.3 Аналіз методів побудови маршрутів

Задача комівояжера – це одна з найпопулярніших задач комбінаторної оптимізації, яка передбачає пошук найвигіднішого маршруту, проходячи хоча б один раз через обрані міста, а потім повертаючись у вихідне місто. При розв’язуванні задачі наводяться: критерій рентабельності маршруту (найкоротший, найдешевший, кумулятивний критерій тощо) та відповідні матриці відстаней, собівартості. Як правило, зазначено, що маршрут повинен проходити через кожне місто лише один раз – у цьому випадку вибір здійснюється за допомогою гамільтонових шляхів. Гамільтонів шлях – це маршрут, який включає кожну вершину графа рівно один раз [6].

Для вирішення цієї задачі вхідні дані можна представити у вигляді математичної моделі. Проблему комівояжера можна представити у вигляді графа, тобто за допомогою вершин і ребер між ними. Вершини графа відповідають містам, а ребра між вершинами – шляхам сполучення між цими містами.

Можна стверджувати, що розв'язання задачі полягає у знаходженні мінімально зваженого гамільтонового шляху в повному зваженому графі. Оскільки комівояжеру необхідно вибрати наступне місто після відвідування кожного з ще не відвіданих міст, існує (n-1)! маршрути для асиметричних і (n-1)!/2 маршрути для симетричної задачі. У цьому випадку розмір пошуку факторно залежить від кількості міст [7].

Розглянемо основні підходи до вирішення задачі комівояжера.

Підхід перебору, або наївний підхід, обчислює та порівнює всі можливі перестановки маршрутів для визначення найкоротшого рішення. Щоб вирішити задачу за допомогою цього підходу, необхідно обчислити загальну кількість маршрутів, а потім намалювати та перерахувати всі можливі маршрути. Розраховуємо відстань кожного маршруту, а потім вибираємо найкоротший, він і є оптимальним рішенням.

Метод гілок та меж розбиває проблему, яку потрібно розв’язати, на кілька підзадач. Це система для розв’язання серії підзадач, кожна з яких може мати кілька можливих рішень і де рішення, обране для однієї проблеми, може впливати на можливі рішення наступних підзадач. Для вирішення задачі необхідно вибрати початковий вузол, а потім встановити прив’язане дуже велике значення. Далі вибрати найдешевшу арку між невідвідуваним і поточним вузлом, а потім додати відстань до поточної відстані. Повторити процес, поки поточна відстань менше межі. Якщо поточна відстань менша за обмежену, можна закінчувати. Тепер можна додати відстань, щоб межа дорівнювала поточній відстані. Необхідно повторювати цей процес, поки не будуть покриті всі дуги.

Ключ до цього іншого методу – методу найближчого сусіду, полягає в тому, щоб завжди відвідувати найближче місце призначення, а потім повертатися до першого міста, коли відвідуються всі інші міста. Щоб вирішити задачу комівояжера за допомогою цього методу, необхідно обрати випадкове місто, знайти найближче невідвідане місто та перейти туди. Після того, як всі міста відвідано, необхідно повернутися до першого міста.

1.3 Постановка задачі

Було проведено аналіз предметної галузі, існуючих аналогів та методів що можуть використовуватись для вирішення проблеми планування подорожі.

Таким чином, мета дослідження полягає в тому, щоб розробити алгоритм, який забезпечував би можливість пропонувати користувачеві міста для відвідування за час поїздки на основі індивідуальних критеріїв користувача. Чим більша точність потрібна користувачеві, тим більше параметрів потрібно буде ввести в систему.

При розробці алгоритму треба використовувати засоби багатокритеріальної оптимізації, зокрема лінійна адитивна згортка з нормуючими множниками та

ваговими коефіцієнтами. Вагові коефіцієнти потрібні щоб враховувати побажання користувача. Також, так як усі дані у різних одиницях вимірювання – нормалізація даних повинна бути проведена. Адитивна згортка обрана замість мультиплікативної адже погане значення одного з критеріїв не повинне дискредитувати увесь варіант подорожі [8].

За допомогою лінійної згортки повинно бути проведене ранжування країн, міст та фінальних комбінацій.

Критеріїв оптимізації може бути нескінченно багато, але в даній роботі на основі експертної оцінки робітників туристичних агентств, було обрано наступні критерії (або фактори), позначивши їх буквою f та номером фактору послідовно:

- f1 – середня температура в місяці обраним користувачем (°C);
- f2 – кількість опадів у місяці обраним користувачем (мм);
- f3 – наявність моря (у наявності 1/ відсутнє 0);
- f4 – наявність гір (у наявності 1/ відсутні 0);
- f5 – ціна за один день (Євро).

За принцип оптимізації виберемо мінімізацію та опишемо алгебраїчний запис лінійної згортки, що повинна бути використана:

$$F = w_1 f_1 + w_2 f_2 + w_3 f_3 + w_4 f_4 + w_5 f_5 \rightarrow min \quad (11)$$

де F – цільова функція;

w – вага відповідного фактору задана користувачем;

f – значення відповідного фактору для конкретного міста або країни.

Значення критеріїв для кожної країни та міста об’єктивні та будуть взятими з мережі Інтернет. Кількість опадів та середню температуру необхідно взяти з ресурсу https://www.weather-atlas.com, ціну за день з ресурсу https://www.budgetyourtrip.com, наявність моря та гір з https://www.google.com/maps або https://www.wikipedia.org/.

Також необхідно використати комбінаторику для створення комбінацій. А для знаходження найвигіднішого шляху використаємо наївний підхід вирішення задачі комівояжера на основі дистанції та/або вартості подорожі.

Після розробки алгоритму необхідно розробити прототип застосунку, який дозволить формувати пропозиції з переліком міст для відвідування на основі побажань користувача [9].

Застосунок має реалізовувати наступний функціонал:

– можливість введення побажань користувача;
– можливість задати важливість кожного побажання;
– формування та відображення пропозицій поїздок.

Для реалізації застосунку слід застосувати мову програмування Kotlin. Користувацьким інтерфейсом повинен бути мобільний застосунок на платформі Android, а отже необхідно використати середу розробки Android Studio.

2 СТВОРЕННЯ АЛГОРИТМУ ПОБУДОВИ МАРШРУТІВ З УРАХУВАННЯМ ПОБАЖАНЬ КОРИСТУВАЧІВ

У попередньому розділі було проаналізовано існуючі методи та виявлено їх специфіку та сформовано постановку задачі, в результаті чого пропонується новий метод, який ґрунтується на комбінації деяких із зазначених методів.

Першим кроком алгоритму є відбір країн та міст на основі побажань користувача. Перелік вподобань може стосуватись безлічі параметрів. Чим більше категорій налаштовано користувачем, тим більш точно буде запропоновано подорож.

По-перше, треба вибрати можливі країни для відвідування користувача. Кількість країн для відвідування в рамках поїздки встановлюється користувачем. Є багато критеріїв, які необхідно враховувати: кількість днів перебування в кожному місті, максимальна загальна вартість, бажані частини світу для відпочинку, наявність моря/океану/річки/воду чи, можливо, важливими є гори, місця орієнтовані на відпочинок, доступний громадський транспорт, клімат, заходи тощо.

На цьому етапі повинна бути використана лінійна адитивна згортка з нормуючими множниками та ваговими коефіцієнтами задля вибору кращих країн. Кроки використання наступні: зводимо дані до одного принципу оптимізації – мінімізації, визначаємо множину Парето, проводимо нормування даних за принципом максиміну, проводимо адитивну згортку помножаючи значення критеріїв на їх вагу, яка встановлена користувачем.

Використовуючи такий самий підхід, необхідно обрати міста в кожній країні.

У результаті першого кроку отримано потенційний список країн та міст. Список відсортований за рейтингом отриманим в результаті проведення операцій згорток відповідно до запитів користувача.

Другим кроком є об’єднання країн у комбінації. Спочатку об’єднаємо країни, комбінації повинні бути рівні розміру який задано користувачем. Для цього

потрібно по-перше, визначити усі можливі комбінації кількості міст у країнах, для цього отримаємо максимальну можливу кількість міст у країні, що буде дорівнювати результату віднімання кількості міст від кількості країн та додання одиниці, далі проведемо операцію перестановок з повторенням за формулою (7) на основі можливої кількості міст та відкинемо опції де сума міст не дорівнює заданій користувачем. По-друге, створимо комбінації країн за допомогою сполучень з комбінаторики за формулою (9). По-трете, створимо комбінації міст в середині країн також за допомогою сполучень з комбінаторики. Останнім кроком проведемо декартове множення міст різних країн для отримання комбінацій міст між країнами.

Результатом другого кроку є варіанти комбінацій міст для відвідування у рамках однієї подорожі.

Третім кроком є встановлення порядку відвідуваності міст для кожної комбінації, що буде базуватись на віддаленості міст одне від одного і/або скільки приблизно буде коштувати переїзд. Для знаходження маршрутів було обрано наївний підхід до вирішення задачі комівояжера. На основі чого обирати маршрут залишається на розсуд користувача – чи то зменшення відстані, чи ціні, чи обох. Але за замовчування краще враховувати обидва параметри – і ціну, і відстань. Наприклад, від Харкова до Відня і до Мілана однакова відстань, але вартість перельоту різна. І навпаки, в іншому прикладі відстань від Мілана до Відня та Венеції різна, але ціна однакова. Якщо вартість розглядається як більш важлива – відстань буде неправильно оброблена, і маршрут може бути зайвим. І якщо відстань розглядається як важливіша за вартість, то загальна вартість поїздки може перевищити очікування. Отже, обидва значення повинні бути нормовані та додані.

Результатом третього кроку є кращі варіанти напрямків для кожної комбінації.

Четвертим і останнім кроком є оцінка всіх підготовлених комбінацій з нормування. Отже, знаходимо суму рейтингів міст в комбінації, які було знайдено на першому кроці, нормалізуємо ці дані, також нормалізуємо ваги маршрутів отримані на попередньому кроці. Множимо значення сумарного рейтингу на 0.7, а

ваги маршруту на 0.3, адже сумарний рейтинг більш важливий, знаходимо суму двох отриманих значень та на їх основі сортуємо дані. Результатом останнього кроку є рейтинг варіантів, а найкращі з них будуть представлені користувачеві.

Також необхідно відмітити, що спеціальним випадком є ситуація коли тільки одне місто потрібне бути обраним. В такому разі тільки перший крок алгоритму буде виконано.

Створимо діаграму активності [10] щоб відобразити логіку запропонованого алгоритму (див. рис. 2.1).

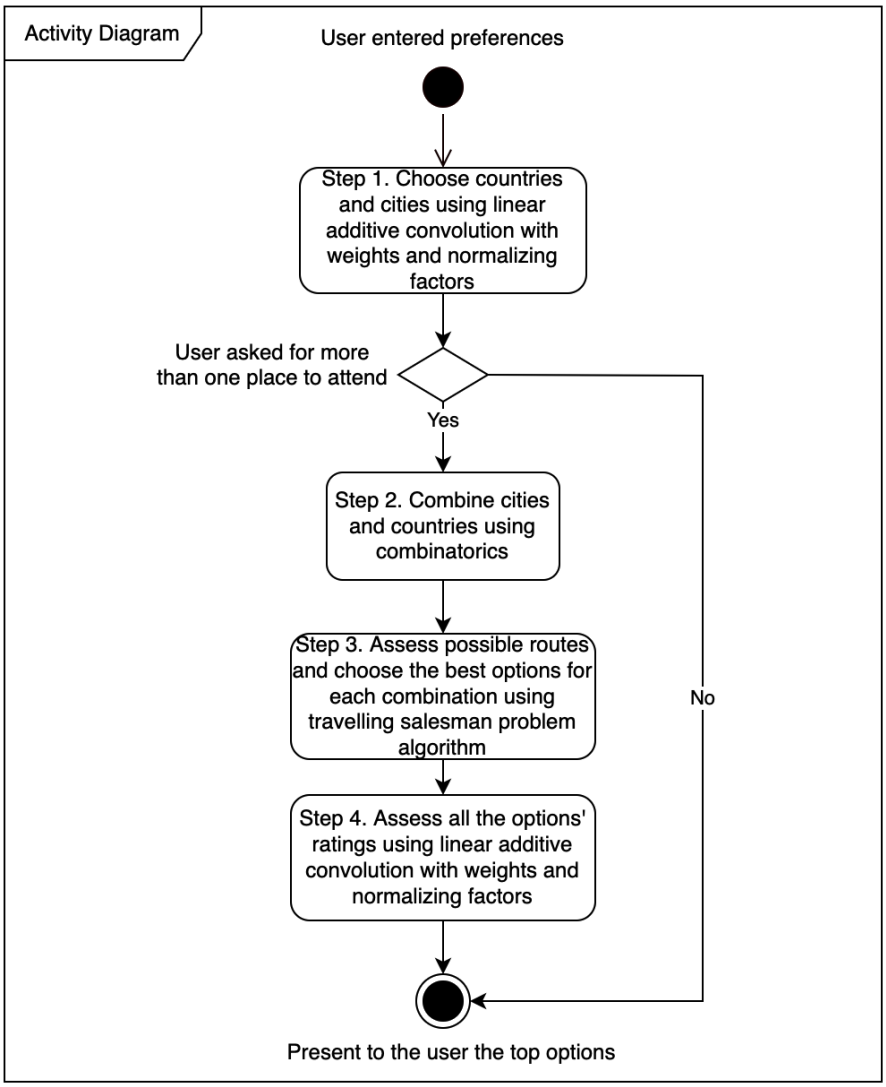

Рисунок 2.1 – Діаграма активності що відображає логіку алгоритму

Отже, було представлено алгоритм для вирішення заданої проблеми планування подорожей [11].

3 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТУ

Згідно встановленим у розділі 2 крокам алгоритму проведемо експеримент. Однак сама задана ситуація і деякі кроки будуть спрощені для демонстраційних цілей.

Припустимо, що користувач бажає відвідати дві країни, по одному місту в кожній, та задає певні побажання. Критерії відбору описані у розділі 1.3. Припустимо, що користувач визначив наступні ваги для заданих критеріїв:

- f1– середня температура в місяці обраним користувачем – 0.2;
- f2 – кількість опадів у місяці обраним користувачем – 0.1;
- f3 – наявність моря – 0.3;
- f4 – наявність гір – 0.2;
- f5 – ціна за один день – 0.2.

Отримуємо векторний опис задачі:

$$w(x) = 0{,}2f1 + 0{,}1f2 + 0{,}3f3 + 0{,}2f4 + 0{,}2f5 \quad (12)$$

Отже, першим кроком виберемо можливі країни для відвідування користувача. Опишемо набір альтернатив: Франція, Італія, Чехія, Німеччина, Іспанія. Занесемо значення критеріїв для кожної країни у таблицю 3.1 базуючись на даних з ресурсів зазначених під час постановки задачі у розділі 1.3.

Таблиця 3.1 – Значення критеріїв для кожної країни

	Температура	Кількість опадів	Наявність моря	Наявність гір	Ціна за день
Франція	26	60	1	1	65
Італія	31	24	1	1	80
Чехія	26	85	0	0	22
Німеччина	24	81	1	0	32
Іспанія	29	27	1	1	50

Наведемо векторний опис і знайдемо найкраще рішення. Спочатку зведемо до одного принципу оптимізації – мінімізації або максимізації. Таким чином температуру, вартість і кількість опадів слід мінімізувати. Тож змінимо значення параметрів наявності моря та гір (див. табл. 3.2).

Таблиця 3.2 – Приведені значення до одного принципу оптимізації

	Температура	Кількість опадів	Наявність моря	Наявність гір	Ціна за день
Франція	26	60	0	0	65
Італія	31	24	0	0	80
Чехія	26	85	1	1	22
Німеччина	24	81	0	1	32
Іспанія	29	27	0	0	50

Визначимо множину Парето. Усі альтернативи не гірші за інші хоча б за одним критерієм, тобто жодна альтернатива не може бути видалена.

Перейдемо до нормалізації критеріїв з урахуванням мінімуму та максимуму на основі наступної формули:

$$x_{normalized} = \frac{x - \min(x)}{\max(x) - \min(x)}, \quad (13)$$

де x – значення що зараз обробляється,

min(x) – мінімальне значення серед усіх,

max(x) – максимальне значення серед усіх [12].

Результати розрахунків зображено в таблиці 3.3.

Таблиця 3.3 – Нормалізовані дані

	Температура	Кількість опадів	Наявність моря	Наявність гір	Ціна за день
Франція	0.29	0.59	0	0	0.74

Продовження таблиці 3.3

	Температура	Кількість опадів	Наявність моря	Наявність гір	Ціна за день
Італія	1	0	0	0	1
Чехія	0.29	1	1	1	0
Німеччина	0	0.93	0	1	0.17
Іспанія	0.71	0.05	0	0	0.48

Виконаємо розрахунки за формулою (12) і зазначено отримані результати в таблиці 3.4.

Таблиця 3.4 – Результати розрахунків

Франція	0.2*0.29 + 0.1*0.59 + 0.3*0 + 0.2*0 + 0.2*0.74 = 0.265
Італія	0.2*1 + 0.1*0 + 0.3*0 + 0.2*0 + 0.2*1 = 0.4
Чехія	0.2*0.29 + 0.1*1 + 0.3*1 + 0.2*1 + 0.2*0 = 0.658
Німеччина	0.2*0 + 0.1*0.93 + 0.3*0 + 0.2*1 + 0.2*0.17=0.327
Іспанія	0.2*0.71 + 0.1*0.05 + 0.3*0 + 0.2*0 + 0.2*0.48=0.243

В результаті Іспанія та Франція мають кращий рейтинг, тому для цього показового експерименту вибрали саме ці країни. Дотримуючись тих самих вищезгаданих кроків, обираються міста в кожній країні: Рейтинг іспанських міст впорядкований за спаданням рейтингу наступний:

– Барселона;
– Валенсія;
– Гранада.

Рейтинг міст Франції впорядкований за спаданням рейтингу наступний:

– Марсель;
– Монпельє;
– Гранада.

Переходимо до другого кроку запропонованого алгоритму, який об’єднує країни в необхідну для користувача кількість країн для відвідування. Оскільки в

даному випадку для демонстраційних цілей було обрано лише дві країни, доступна лише одна комбінація – Іспанія та Франція. Але комбінації для обраних міст потрібно створити. У поточному експерименті користувач запросив по одному місту з кожної з двох країн.

Наразі з кожної країни обираються по три міста, потім у результаті буде отримано дев’ять комбінацій за допомогою множення комбінацій. Тоді результат зі списком комбінацій міст такий: Барселона і Марсель, Барселона і Монпельє, Барселона і Ніцца, Валенсія і Марсель, Валенсія і Монпельє, Валенсія і Ніцца, Гранада і Марсель, Гранада і Монпельє, Гранада і Ніцца.

Третій крок – вирішення проблеми комівояжера. Спочатку необхідно надати значення цін і відстаней в обох напрямках.

Таблиця 3.5 заповнена відстанями між містами, які можна ввести в один маршрут на основі середніх значень з Інтернету. Значення міст, які не можуть бути в одному маршруті в поточному випадку, наприклад, Валенсія та Барселона опущені.

Таблиця 3.5 – Відстані між містами

	Харків	Барселона	Валенсія	Гранада	Марсель	Монпельє	Ніцца
Харків	–	3602	3944	4448	3277	3267	2871
Барселона	3602	–	–	–	507	348	664
Валенсія	3944	–	–	–	846	687	1004
Гранада	4448	–	–	–	1350	1192	1508
Марсель	3277	507	846	1350	–	–	–
Монпельє	3267	348	687	1192	–	–	–
Ніцца	2871	664	1004	1508	–	–	–

Потім значення відстані між містами необхідно нормалізувати, щоб відстані можна було правильно співвіднести з цінами. Результати представлені в таблиці 3.6.

Таблиця 3.6 – Нормалізовані значення відстаней

	Харків	Барселона	Валенсія	Гранада	Марсель	Монпельє	Ніцца
Харків	–	0.8	0.87	1	0.71	0.71	0.6
Барселона	0.8	–	–	–	0.04	0	0.08
Валенсія	0.87	–	–	–	0.12	0.08	0.16
Гранада	1	–	–	–	0.24	0.2	0.28
Марсель	0.71	0.04	0.12	0.24	–	–	–
Монпельє	0.71	0	0.08	0.2	–	–	–
Ніцца	0.6	0.08	0.16	0.28	–	–	–

Ціни на поїздку з одного міста в інше відрізняються залежно від напрямку. Значення відображені в таблиці 3.7 на основі середніх даних з Інтернету.

Таблиця 3.7 – Ціна дороги між містами

	Харків	Барселона	Валенсія	Гранада	Марсель	Монпельє	Ніцца
Харків	–	75	90	124	83	128	82
Барселона	71	–	–	–	37	75	26
Валенсія	85	–	–	–	21	78	71
Гранада	114	–	–	–	64	113	56
Марсель	75	30	17	56	–	–	–
Монпельє	120	67	68	106	–	–	–
Ніцца	78	22	60	48	–	–	–

Як і для відстаней, ціни також потрібно нормалізувати, щоб їх можна було правильно співвідносити з відстанями. Результати представлені в таблиці 3.8.

Таблиця 3.8 – Нормалізовані значення цін

	Харків	Барселона	Валенсія	Гранада	Марсель	Монпельє	Ніцца
Харків	–	0.45	0.57	0.84	0.52	1	0.5
Барселона	0.42	–	–	–	0.16	0.45	0.07
Валенсія	0.53	–	–	–	0.03	0.47	0.42
Гранада	0.76	–	–	–	0.37	0.75	0.3

Продовження таблиці 3.8

	Харків	Барселона	Валенсія	Гранада	Марсель	Монпельє	Ніцца
Марсель	0.45	0.1	0	0.3	–	–	–
Монпельє	0.8	0.4	0.4	0.7	–	–	–
Ніцца	0.48	0.04	0.34	0.24	–	–	–

Значення суми нормованих значень ціни та відстані відображені в таблиці 3.9.

Таблиця 3.9 – Сума нормованих значень ціни та відстані

	Харків	Барселона	Валенсія	Гранада	Марсель	Монпельє	Ніцца
Харків	–	1.25	1.44	1.84	1.23	1.71	1.1
Барселона	1.22	–	–	–	0.2	0.45	0.15
Валенсія	1.4	–	–	–	0.15	0.55	0.58
Гранада	1.76	–	–	–	0.61	0.95	0.58
Марсель	1.16	0.14	0.12	0.54	–	–	–
Монпельє	1.51	0.4	0.48	0.9	–	–	–
Ніцца	1.08	0.12	0.5	0.62	–	–	–

На основі значень для міст Харкова, Марселя та Барселони створено наступний графік для демонстрації варіантів (див. рис. 3.1).

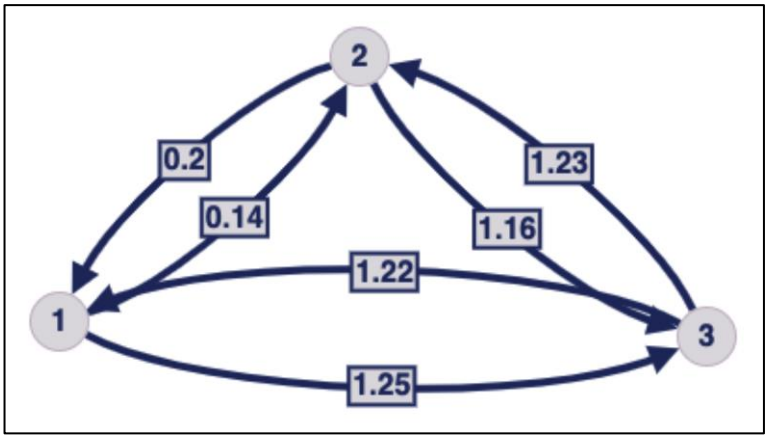

Рисунок 3.1 – Граф маршрутів Харків (1), Марсель (2) і Барселона (3) та їх ваги

На основі значень із таблиці 3.9 вирішуємо задачу комівояжера та отримуємо наступний список найкращих маршрутів для кожної комбінації:

– П1: Харків – Марсель – Барселона – Харків з вагою 2.52;
– П2: Харків – Барселона – Монпельє – Харків з вагою 3.21;
– П3: Харків – Ніцца – Барселона– Харків з вагою 2.44;
– П4: Харків – Марсель – Валенсія– Харків з вагою 2.75;
– П5: Харків – Валенсія – Монпельє– Харків з вагою 3.5;
– П6: Харків – Ніцца – Валенсія– Харків з вагою 3;
– П7: Харків – Марсель – Гранада– Харків з вагою 3.53;
– П8: Харків – Гранада – Монпельє– Харків з вагою 4.3;
– П9: Харків – Ніцца – Гранада– Харків з вагою 3.48.

Четвертим і останнім кроком алгоритму є оцінка всіх підготовлених комбінацій. Буде використана лінійна адитивна згортка з нормуючими множниками. Вага не розглядається, оскільки всі три критерії – рейтинг задачі комівояжера, рейтинг іспанського міста, рейтинг французького міста – однаково важливі. Збираємо всі оцінки найкращих варіантів із попереднього кроку та помістимо їх у стовпець оцінки проблем комівояжера. Рейтинги міст беруться з першого кроку алгоритму. Усі перераховані рейтинги відображені в таблиці 3.10.

Таблиця 3.10 – Рейтинги отриманих комбінацій

	Ціна маршруту	Рейтинг міста з Іспанії	Рейтинг міста з Франції
П1	2.52	1	1
П2	3.21	1	2
П3	2.44	1	3
П4	2.75	2	1
П5	3.5	2	2
П6	3	2	3
П7	3.53	3	1
П8	4.3	3	2
П9	3.48	3	3

У цьому випадку всі дані зводяться до мінімуму, тому на даний момент коригувати дані не потрібно. Потім потрібно нормалізувати значення за допомогою формули (13). Результати відображені в таблиці 3.11.

Таблиця 3.11 – Нормалізація значень

	Вартість	Рейтинг міста з Іспанії	Рейтинг міста з Франції
П1	0.04	0	0
П2	0.41	0	0.5
П3	0	0	1
П4	0.16	0.5	0
П5	0.57	0.5	0.5
П6	0.3	0.5	1
П7	0.59	1	0
П8	1	1	0.5
П9	0.56	1	1

Знайдемо суму рейтингів міст для кожної опції подорожі, помножимо її на 0.7, а вартість маршруту на 0.3, адже сума рейтингів важливіша. Додамо отримані два значення та отримаємо наступні результати: П1 – 0.012, П2 – 0.473, П3 – 0.7, П4 – 0.398, П5 – 0.871, П6 – 1.14, П7 – 0.872, П8 – 1.35, П9 – 1.568.

Таким чином найкращими варіантами є:

- П1: Харків – Марсель – Барселона – Харків;
- П4: Харків – Марсель – Валенсія– Харків;
- П2: Харків – Барселона – Монпельє – Харків.

Отже, було проведено експеримент виконуючи кроки запропонованого алгоритму.

4 ПРОЕКТУВАННЯ СИСТЕМИ

4.1 Проектування реалізації алгоритму

Щоб створити масштабовану, надійну і легко підтримувану систему, необхідно використовувати модульну конструкцію. Отже необхідно розділити обов’язки між компонентами, які реалізують кроки алгоритму. Одною з переваг є проста заміна або покращення алгоритмів в окремих кроках при розвиненні системи. А також покращуються можливість юніт тестування.

Діаграма послідовності [13] (див. рис. 4.1) демонструє послідовність дій в системі з окремими модулями при плануванні маршрутів.

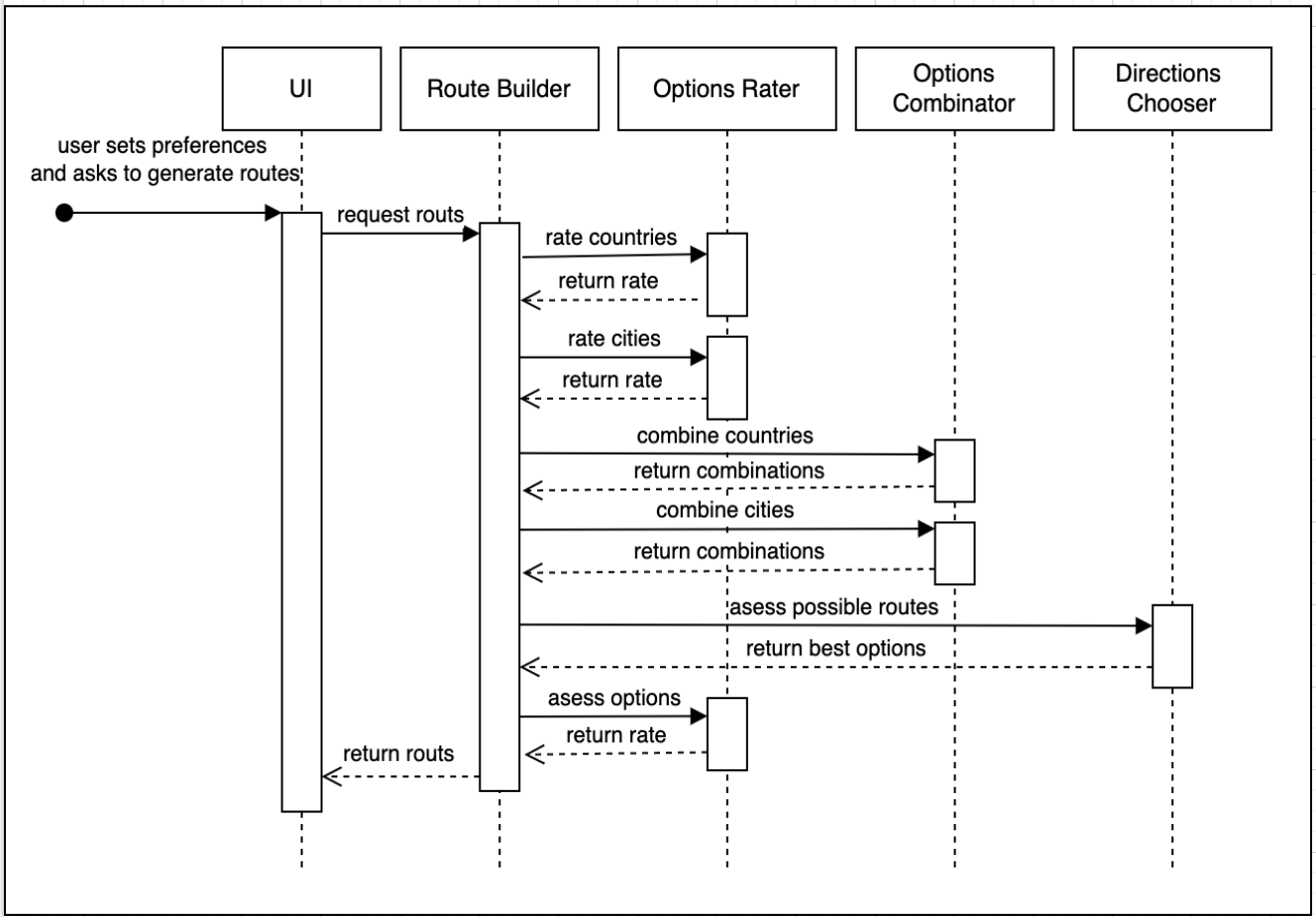

Рисунок 4.1 – Діаграма послідовності роботи алгоритму

Кожен з модулей зображених на діаграмі відповідає за певну логічну частину алгоритму.

Модуль «UI» відповідає відображення інформації системи та за комунікацію з користувачем. Отже, він отримує дані про вподобання користувача у деяких категоріях та при отриманні запиту користувача на планування подорожі передає отримані дані до модулей алгоритму.

Модуль «Route Builder» це головний модуль, який спілкується з іншими. Він отримує дані щодо вподобань користувача та по черзі викликає необхідні модулі задовольняючи логіку алгоритму.

Модуль «Options Rater» займається оцінкою країн, міст, опцій та іншого. Наразі його реалізацію повинна бути виконана як лінійна адитивна згортка з нормуючими множниками та ваговими коефіцієнтами.

Модуль «Options Combinator» створює комбінації країн та міст у них, отже відповідає за комбінаторні задачі.

Модуль «Directions Chooser» відповідає за вирішення напрямків слідування для конкретних комбінацій міст. Наразі його реалізація повинна містити вирішення задачі комівояжера.

Опишемо комунікацію модулів за діаграмою. Користувач, який встановлює параметри та просить створити маршрут, запускає виконання алгоритму, дані передаються з «UI» модуля до «Route Builder». Даний модуль в свою чергу спочатку звертається до «Options Rater» для оцінки країн і міст, потім варіанти об’єднуються за допомогою «Options Combinator», а вибираються найкращі напрямки в кожній комбінації за допомогою модуля «Directions Chooser». Як останній крок перед показом результату користувачеві, «Options Rater» викликається ще раз з урахуванням усіх попередніх ставок [14].

Отже, модульність системи дозволить легко модифікувати, додавати та видаляти кроки алгоритму за необхідності при розвиненні системи, а також окремі алгоритму зможуть бути якісно протестовані.

4.2 Проектування архітектури проекту

По мірі того як програми Android збільшуються, важливо визначити архітектуру, яка дозволить програмі масштабуватися, підвищить її надійність і полегшить тестування програми. Архітектура програми визначає межі між частинами програми та обов'язки, які має мати кожна частина. Найважливішим принципом, якого слід дотримуватися, є розділення відповідальностей. Поширеною помилкою є написання всього коду в Activity або Fragment. Ці класи на основі інтерфейсу користувача повинні містити лише логіку, яка обробляє взаємодію користувача та операційної системи. Зберігаючи ці класи якомога простими, можна уникнути багатьох проблем, пов'язаних із життєвим циклом компонентів, і покращити можливість тестування цих класів [17].

Розглянемо запропоновані до використання шари чистої архітектури та види класів що до них належать.

Існує три основні шари – доменний, шар даних та презентаційний. Розглянемо їх докладніше.

Перший шар презентаційний. Роль цього рівня полягає у відображенні даних програми на екрані. Щоразу, коли дані змінюються через взаємодію з користувачем, наприклад, натискання кнопки, або зовнішній вхід, наприклад, відповідь мережі, інтерфейс користувача має оновлюватися, щоб відобразити зміни.

Наступний шар це доменний. Він містить сутності та бізнес правила предметної галузі, в тому числі розрахунки.

Сутності інкапсулюють бізнес-правила. Сутність може бути об'єктом з методами, або це може бути набір структур даних і функцій. Випадки використання містять конкретні бізнес-правила програми. Цей шар інкапсулює та реалізує всі варіанти використання системи. Ці варіанти використання організовують потік даних до сутностей і від них і спрямовують ці об'єкти використовувати свої бізнес-правила в масштабі підприємства для досягнення цілей варіанта використання [18].

Також цей шар містить інтерфейс репозиторію для доступу до даних, таким чином він може бути використаний у випадках використання.

Останній шар доступу до даних. Він містить в собі реалізацію інтерфейсу репозиторія в якій використовує джерела даних для їх отримання або оновлення.

Доменний не повинен залежати від інших шарів, він отримує запити від презентаційного шару та за допомогою отриманих даних через інтерфейс репозиторію з шару даних, повертає дані до презентаційного шару. Шар даних та презентаційний повинні залежати тільки від доменного шару, бізнес-логіки, та не повинні знати один про одного (див. рис. 4.2).

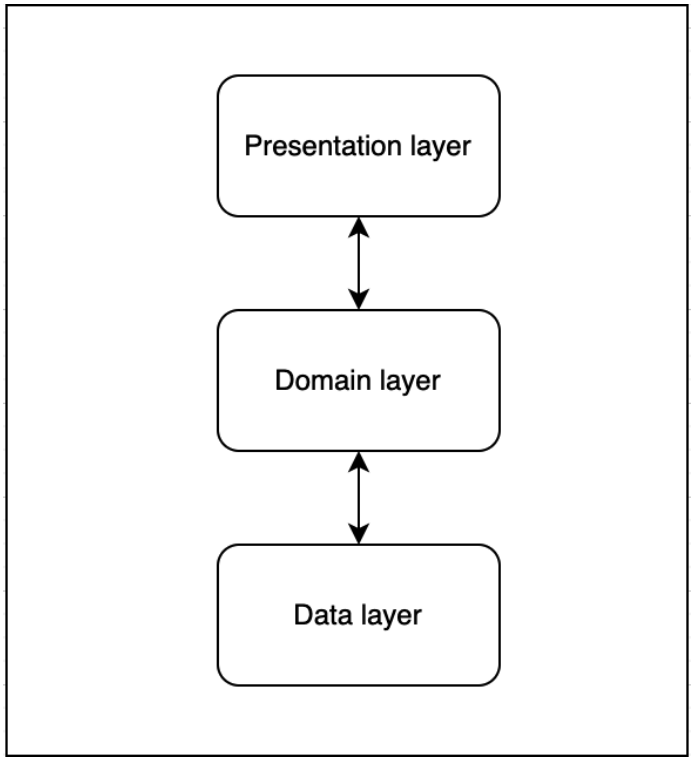

Рисунок 4.2 – Тришарова архітектура

Також для чистої архітектури та коду корисним є використання принципу ін’єкції залежностей – це техніка, яка широко використовується в програмуванні і добре підходить для розробки Android. За допомогою цього підходу залежності класу надаються йому ззовні, а не екземпляр класу створює або отримує їх сам. Реалізація ін’єкції залежностей надає наступні переваги: простота рефакторингу, простота тестування, можливість повторного використання коду [19].

5 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

5.1 Опис засобів розробки

Під час постановки задачі було позначено що необхідно реалізувати програмний користувацький інтерфейс у вигляді мобільного застосунку під платформу Android.

Середовище розробки Android Studio надає безліч зручних додатків та можливостей для розробки мобільного застосунку, тому і буде використано.

Для написання алгоритму та інтерфейсу буде застосовано мову програмування Kotlin. У 2019 році на конференції Google I/O, Kotlin було визнано кращим варіантом для розробки під Android.

Kotlin – це статично-типізована мова програмування, яка дозволяє писати зрозумілий та стислий код. Зазначимо головні переваги Kotlin:

– краща читабельність та менша кількість коду, витрачається менше часу на його написання та на спроби зрозуміти код, що написаний іншими;
– зріла мова та середовище. З 2011 року Kotlin постійно розвивається. Зараз він інтегрований в Android Studio і використовується багатьма компаніями для розробки застосунків під Android;
– підтримка Kotlin в Android Jetpack. Існуючі бібліотеки Android розширюють з використанням функцій даною мови для більш зручної роботи;
– взаємодія з Java. Є можливість використовувати Kotlin разом із Java у програмах без перенесення всього коду на Kotlin, а отже можна використовувати і безліч існуючих бібліотек, що були написані на Java [15].

Щодо сторонніх бібліотек, CombinatoricsKt [16] буде використано для вирішення комбінаторних завдань.

5.2 Опис реалізованої архітектури мобільного застосунку

Реалізуємо чисту архітектуру слідуючи принципам з розділу 4.2

На рисунку 5.1 зображено знімок екрану з ієрархією класів доменного шару у проекті реалізованого застосунку. Бачимо пакет з варіантами використання – GetCities, GetSuggestions, GetWeights. Також інтерфейс репозиторію – CountriesRepository. Файл, що містить усі сутності бізнес-логіки – Model. Та файл з методами алгоритму у пакеті algorithm.

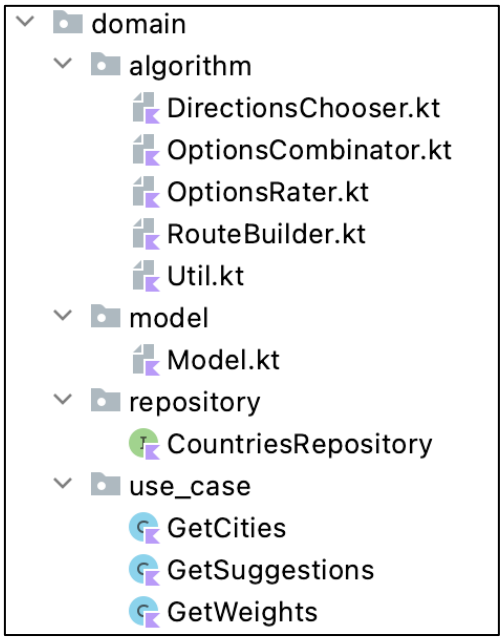

Рисунок 5.1 – Доменний шар

Шар доступу до даних містить реалізацію інтерфейсу репозиторія – CountriesRepositoryImpl, та джерело даних DataSource, використовується в репозиторії (див. рис. 5.2).

Рисунок 5.2 – Шар даних

Презентаційний шар містить Activity, ViewModel та Adapter класи для керування логікою відображення інформації для усіх трьох екранів – зазначення побажань, вибір пріоритетів та відображення пропозицій (див. рис. 5.3).

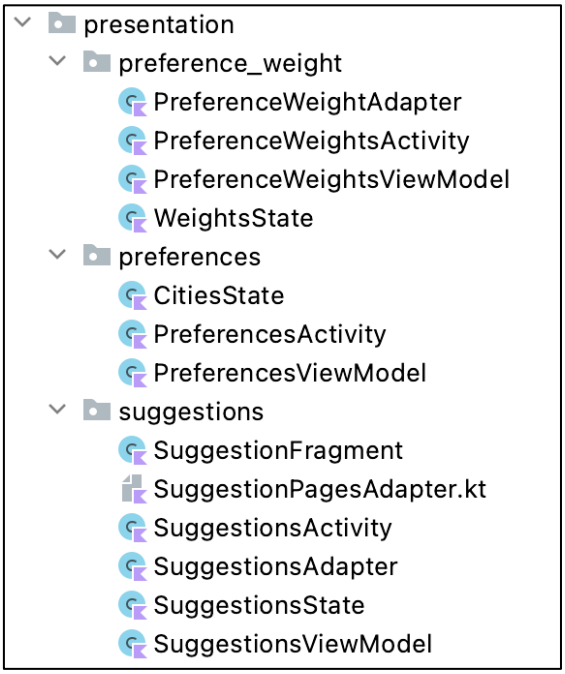

Рисунок 5.3 – Презентаційний шар

Отже, при реалізації застосунку було використано чисту тришарову архітектуру.

5.3 Опис програмної реалізації алгоритму

Реалізуємо алгоритм розділивши за відповідальністю методи на окремі файли як було описано у розділі 4.

У файлі RouteBuilder (див. додаток А) напишемо основний метод алгоритму, який викликатиме необхідні методи з інших файлів, що задовольняють зазначеній раніше логіці алгоритму:

```
    val countries = chooseCountry(countriesList,
allPreferences.countryAmount, allPreferences.preferences)
    val cities = chooseCity(cityList, allPreferences.cityAmount,
allPreferences.preferences, countries).toMutableList()
    val sets =
createSets(allPreferences.cityAmount,allPreferences.countryAmount,
countries, cities)
    val evaluatedSets = evaluateSets(sets, startCity,
allPreferences.routePreference)
    val result = evaluateTheResults(evaluatedSets)
```

Розглянемо частини коду файлу OptionsRater (див. додаток Б) який робить оцінку країн, міст та фінальних опцій. Для оцінки країн та міст використовується лінійна адитивна згортка з нормуючими множниками та ваговими коефіцієнтами.

Спочатку усі дані обробляються щоб слідувати одному принципу оптимізації, в даному випадку було обрано мінімізацію. Отже, якщо, наприклад, користувач задав як побажання гарячий клімат, то усі дані змінюють значення на їх попереднє значення що було відняте від максимального:

```
    preferences.forEach { pref ->
        when (pref.type) {
            PreferenceType.TEMPERATURE -> {
                    if (pref.value == Climate.HOT.ordinal) {
                        countries.map { it.averageSummerTemperature
}.maxByOrNull { it }?.let { value ->
                        countries.forEach { it.averageSummerTemperature =
value - it.averageSummerTemperature }
                    }
                }
            }
```

Далі перевіряємо множину Парето. Шукаємо чи є якась країна, яка гірша гірші за інші по всім критеріям. Покажемо пошук найгірших опцій на прикладі з температурою:

```
    val countriesWithLeastPreferredValues = mutableListOf<String>()
    preferences.forEach { pref ->
            when (pref.type) {
                PreferenceType.TEMPERATURE -> {
                    val temperature = when (pref.value) {
                        Climate.HOT.ordinal -> countries.map {
it.averageSummerTemperature }.maxByOrNull { it }
                        Climate.COLD.ordinal -> countries.map {
it.averageSummerTemperature }.minByOrNull { it }
                        else -> null
                    }
                    temperature?.let {
```

```
                        countries.filter { it.averageSummerTemperature ==
temperature }.map { it.name }
                        .let {
countriesWithLeastPreferredValues.addAll(it) }
                }
            }
```

Якщо країна зустрілась у списку найгірших таку кількість раз, скільки було задано побажань, то вона гірша по всім критеріям і її необхідно видалити:

```
    val countriesToBeRemoved =
countriesWithLeastPreferredValues.groupingBy { it }.eachCount().filter {
it.value == preferences.size }.keys
    countriesToBeRemoved.forEach { name ->
countries.remove(countries.find { it.name == name }) }
```

Далі нормалізуємо значення за принципом максиміну:

```
    preferences.forEach { pref ->
    val max: Double? = countries.map { readInstanceProperty<Double>(it,
pref.fieldName) }.maxByOrNull { it }
    val min: Double? = countries.map { readInstanceProperty<Double>(it,
pref.fieldName) }.minByOrNull { it }
    if (max != null && min != null && max != min) {
            countries.forEach {
                val value: Double = ((readInstanceProperty<Double>(it,
pref.fieldName) - min) / (max - min)).toDouble()
                setInstanceProperty(it, pref.fieldName, value)
            }
        }
    }
```

Далі проведемо лінійну згортку враховуючи побажання користувача які враховані за допомогою ваги побажання:

```
    preferences.forEach { pref ->
    countries.forEach {
            it.linearRating += (readInstanceProperty<Double>(it,
pref.fieldName) * pref.weight)
        }
    }
```

Результатом є відсортовані за рейтингом, що є результатом лінійної згортки, значення. Такі ж самі операції виконуються для вибору міст.

Далі алгоритм звертається до файлу OptionsCombinator (див. додаток В) який створює комбінації країн та міст у них.

Спочатку отримаємо можливу кількість міст в країнах. Наприклад, якщо побажав відвідати 2 країни та 3 міста, то отримаємо 1 і 2:

```
    val possibleCitiesAmount = mutableListOf<Int>().apply {(cityAmount -
countryAmount + 1).downTo(0).forEachIndexed { index, _ -> add(index + 1) }}
```

Далі отримаємо можливі комбінації кількостей міст між країнами. Отже для прикладу вище отримаємо [1,2] та [2,1]:

```
    val combCitiesNumbers =
possibleCitiesAmount.permutationsWithRepetition(countryAmount).toList().fil
ter { it.sum() == cityAmount.toList()
```

Отримаємо комбінації країн, а точніше їх індексів:

```
    val countriesCombIndices =
CombinationsGenerator.indices(countriesList.size, countryAmount).toList()
```

Тепер для кожної отриманої комбінації країн створимо можливі комбінації міст:

```
    combCitiesNumbers.forEach { ints ->
            val combinationsWithinCountries =
mutableListOf<List<List<City>>>()
            ints.forEachIndexed { index, i ->
                combinationsWithinCountries.add(
    mapCountryCities[countriesCombination[index]]!!.combinations(i).toLis
t()
                )
            }
```

Далі за допомогою поступового виклику наступного методу здійснюємо перемноження списків комбінацій міст між країнами:

```
    private fun cartesianProduct(firstList: List<List<City>>, secondList:
List<List<City>>): List<List<City>> {
    return firstList.flatMap { firstListElem ->
        secondList.map { secondListElem ->
           listOf(firstListElem, secondListElem).flatten()
        }
    }
```

Перейдемо до файлу DirectionsChooser (див. додаток Г) який відповідає за вибір напрямків слідування для конкретних комбінацій. Його реалізація містить вирішення задачі комівояжера. Робота повинна бути проведена на нормалізованих даних, чи то нормалізовані значення відстаней, чи цін, чи їх сума. Наступний алгоритм створює усі можливі шляхи для заданих комбінацій, а далі знаходить суму їх дистанцій та/або сум і знаходить найменший варіант:

```
val cities: List<String> = distances.keys.toList()
val paths: List<MutableList<String>> = getPermutations(cities)
distances[startCity] = startCityDistances
paths.forEach { it.add(0, startCity) }
var shortestPath = mutableListOf<String>()
var minDistance = Double.MAX_VALUE
for (path in paths) {
    var distance = countPathDistance(path, distances)
    distance += distances[path[path.size - 1]]!![path[0]]!!
    if (distance < minDistance) {
        minDistance = distance
        shortestPath = path
   }
}
shortestPath.add(shortestPath[0])
```

Останнім кроком є оцінка рейтингів отриманих комбінацій. Для оцінки використовується лінійна адитивна згортка з нормалізованими даними. Спочатку знаходимо суму лінійних рейтингів кожного міста з комбінації:

```
val results = sets.map { set ->
    val sum = set.first.sumOf { it.linearRating }
    Combination(set.first, set.second, sum)
}.toList()
```

Далі нормалізуємо дані сумарного рейтингу використовуючи принцип максиміну:

```
val maxRating: Double? = (results.map { it.sumRating }.maxByOrNull {
it })
val minRating: Double? = (results.map { it.sumRating }.minByOrNull {
it })
if (maxRating != null && minRating != null && maxRating != minRating)
{
    results.forEach { it.sumRating = (it.sumRating - minRating) /
(maxRating - minRating) }
}
```

І теж саме для ваги маршруту отриманої на попередньому кроці з вибором маршрутів. Далі для кожного елементу помножуємо рейтинги на коефіцієнти, адже сумарний лінійний рейтинг важливіший за вагу маршруту, отримуємо суму цих значень та фільтруємо список комбінацій:

```
return results.sortedBy { it.sumRating * 0.7 + it.tspRating * 0.3 }
```

Отже, було розроблено алгоритм, що був описаний у попередніх розділах.

5.4 Опис роботи застосунку

Розглянемо роботу розробленого застосунку. На рисунку 5.4 зображено два знімки екранів. Перший – початковий екран де користувач має можливість задати наступні побажання: кількість міст, кількість країн, клімат, опади, наявність гір поблизу, моря, максимальна вартість за день, місто початку подорожі та на основі чого обирати маршрут – вартості чи дистанцій. На другому екрані користувачу потрібно задати пріоритетність побажань перетягуючи їх щоб встановити необхідний порядок – від найбільш важливого до найменш.

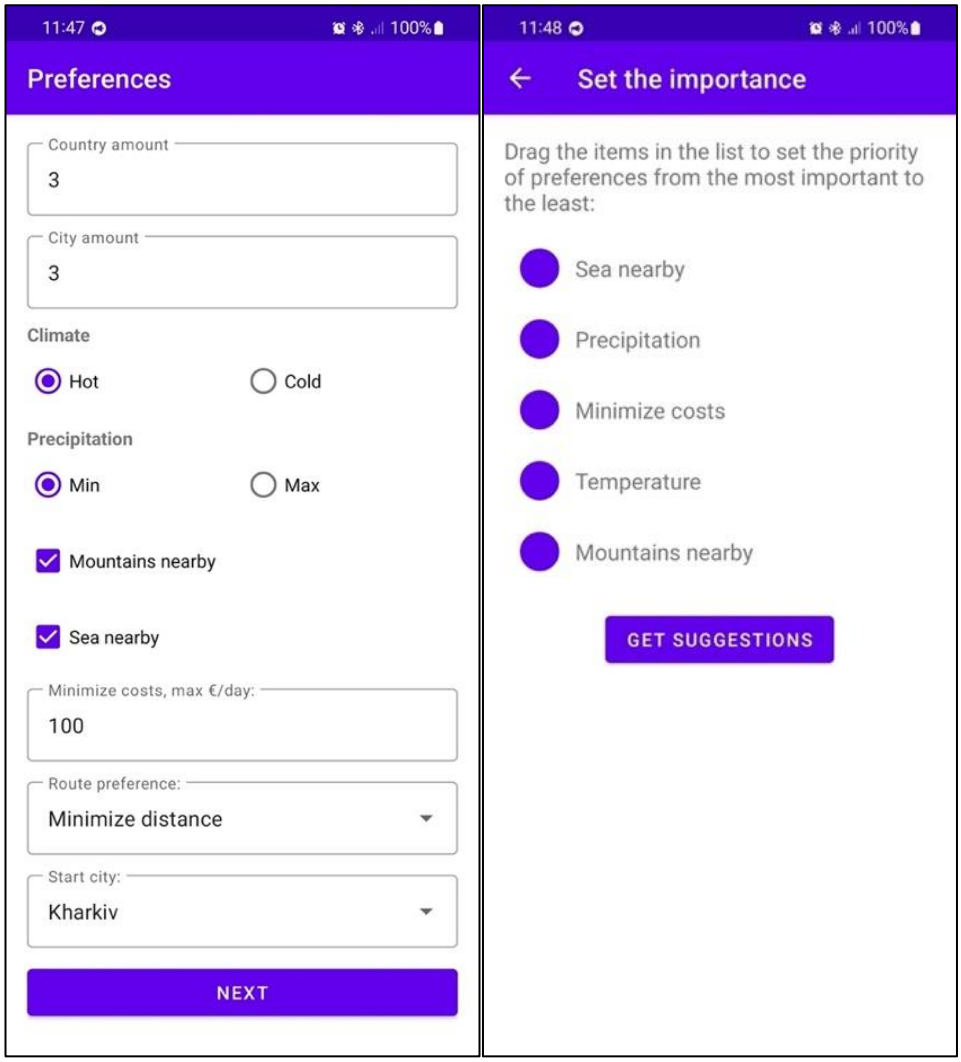

Рисунок 5.4 – Знімки екранів з побажаннями користувача та із встановленням пріоритетів

На наступних екранах користувач отримує п’ять пропозицій на основі введених побажань. На екранах зображено міста, дистанції між ними та вартість за день (див. рис. 5.5)

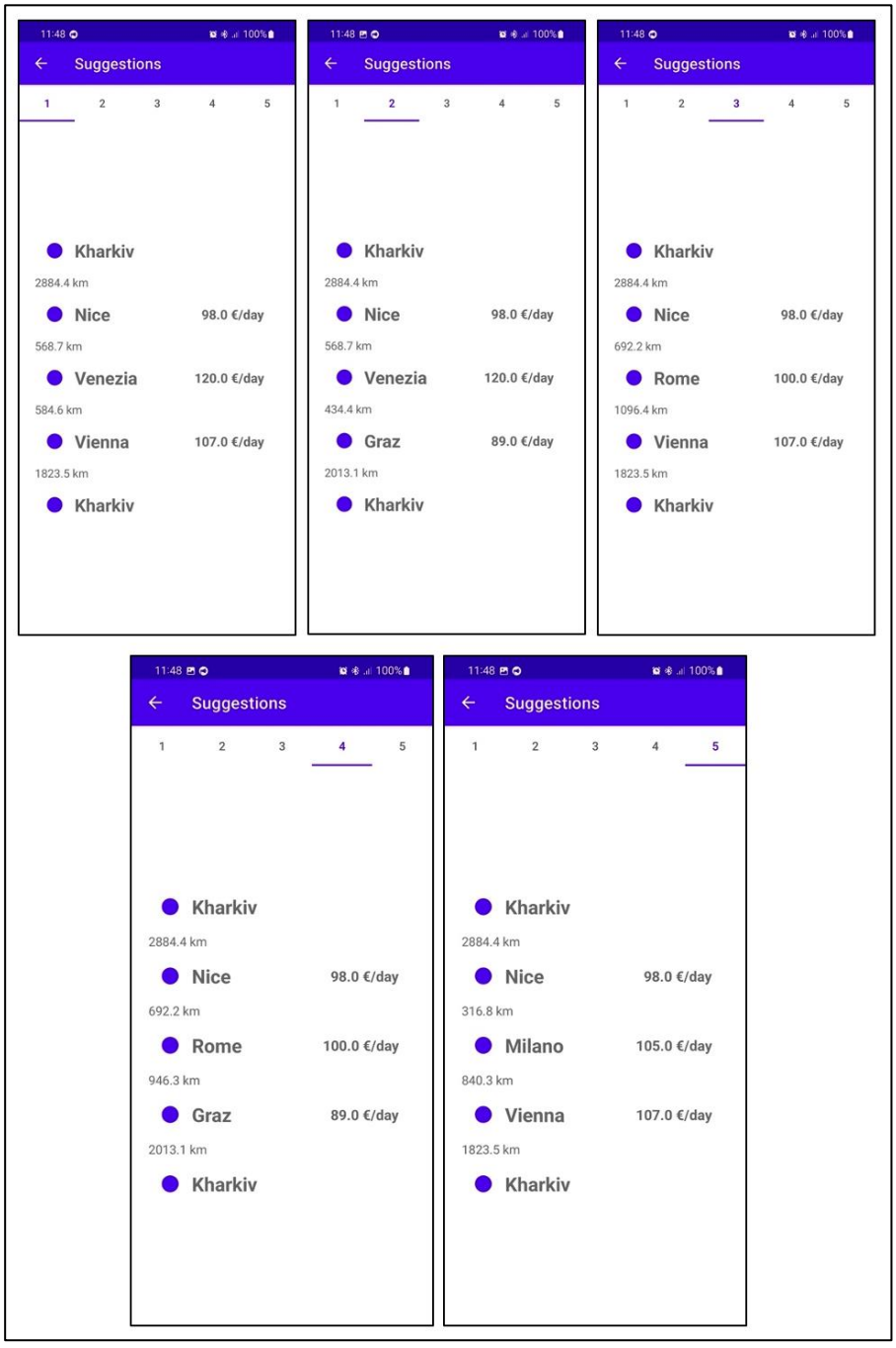

Рисунок 5.5 – Знімки екрану з запропонованими опціями

Продемонструємо також роботи системи з іншими заданими значеннями. Визначимо побажання щодо більш холодного клімату, максимальних опадів, наявність гір та відсутність моря поруч. Введені дані зображено на рисунку 5.6.

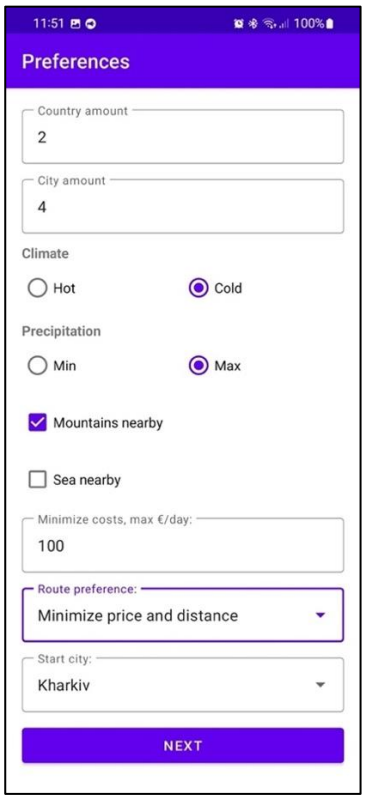

Рисунок 5.6 – Знімок екрану з побажаннями користувача

Також визначимо вищий пріоритет для останніх двох критеріїв (див. рис. 5.7).

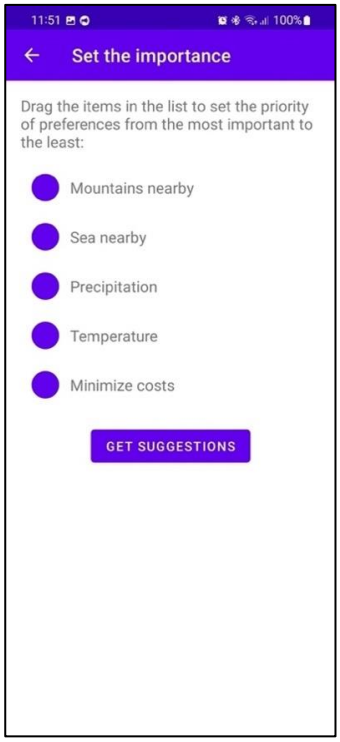

Рисунок 5.7 – Знімок екрану із встановленням пріоритетів

На рисунку 5.8 можем побачити, що алгоритм пропонує міста що задовольняють побажанням користувача.

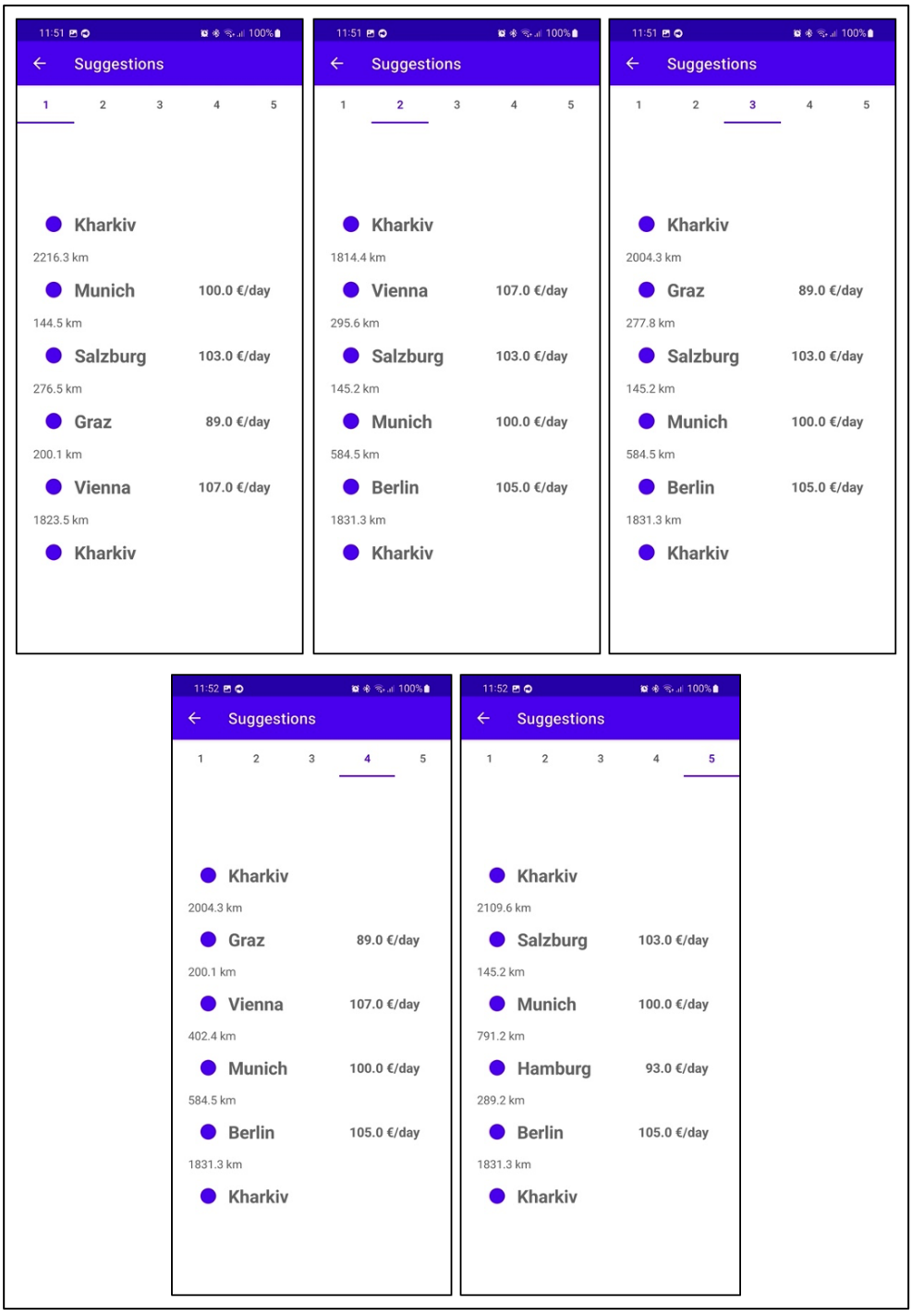

Рисунок 5.8 – Знімки екрану з запропонованими опціями

Отже, було продемонстровано роботу програмної реалізації алгоритму пропонування подорожей.

5.5 Тестування програмної реалізації

5.5.1 Мануальне тестування

Проведемо два види тестування – мануальне тестування та автоматизоване юніт тестування.

Мануальне тестування – це вид тестування програмного забезпечення, коли тести виконуються вручну людиною без використання автоматизованих інструментів. Метою даного тестування є виявлення проблем, помилок, дефектів у програмному застосунку. Даний вид тестування програмного забезпечення є найбільш примітивною технікою з усіх видів тестування.

Будь-яку програму необхідно тестувати вручну. Концепції даного тестування не вимагають знання жодного інструменту тестування. Ключова концепція ручного тестування полягає в тому, щоб переконатися, що дана програма не містить помилок і працює відповідно до функціональних вимог.

Мануальне тестування програмного забезпечення вимагає більше зусиль ніж автоматизоване, але воно необхідне адже автоматизований тест не замінить людського сприйняття. Ручне тестування є важливою частиною розробки зручного програмного забезпечення, адже люди беруть участь у тестуванні програмних застосунків, а кінцеві користувачі також є людьми. Вони повинні діяти з точки зору користувача.

5.5.2 Юніт тестування

Юніт тестування – це спосіб тестування найменшої частини коду, яку можна логічно виділити в системі. У більшості мов програмування це метод, функція, підпрограма, або властивість.

Юніт тестування є важливим кроком у процесі розробки, адже воно може допомогти виявити ранні недоліки в коді, які може бути складніше знайти на пізніших етапах тестування.

Напишемо юніт тести мовою програмування Kotlin за допомогою фреймфорку для тестування JUnit.

Розглянемо перший приклад юніт тесту. Визначимо наступні побажання користувача: побувати у двох країнах, по одному місту, найголовнішим побажанням є наявність моря поруч, далі менша кількість опадів, далі висока температура і найменш важливі мінімізація ціни та гори поруч. Тестовими даними будуть наступні міста: Ніца, Рим, Відень та Берлін. Тож перевіряємо що при вказаних побажаннях алгоритм запропонує Ніцу та Рим:

```
    @Test
    fun test_seaNearby_lowPrecipitation_highTemperature() {
        val allPreferences = AllPreferences(
            2, 2, "Kharkiv",
            RoutePreference.MIN_PRICE_AND_DISTANCE,
            listOf(
                Preference(PreferenceType.SEA_NEARBY, 1,
calculateWeight(1)),
                Preference(PreferenceType.PRECIPITATION, 0,
calculateWeight(2)),
                Preference(PreferenceType.TEMPERATURE, 1,
calculateWeight(3)),
                Preference(PreferenceType.MINIMIZE_COSTS, 110,
calculateWeight(4)),
                Preference(PreferenceType.MOUNTAINS_NEARBY, 1,
calculateWeight(5)),
            )
        )

        val suggestions = getTripSuggestions(data, allPreferences)
        assert(suggestions.isNotEmpty())

        val firstSuggestions = suggestions.first().cities.map { it.name }
        assert(firstSuggestions.contains("Nice"))
        assert(firstSuggestions.contains("Rome"))
    }
```

Розглянемо другий приклад юніт тесту. Визначимо інші побажання користувача: також побувати у двох країнах, по одному місту, але найголовнішим побажанням є наявність гір поруч, далі менша температура, більша кількість опадів

і найменш важливі море поруч та мінімізація цін. Перевіряємо, що при вказаних побажаннях алгоритм запропонує Відень та Берлін:

```
    @Test
    fun test_mountainsNearby_lowTemperature_highPrecipitation() {
        val allPreferences = AllPreferences(
            2, 2, "Kharkiv",
            RoutePreference.MIN_PRICE_AND_DISTANCE,
            listOf(
                Preference(PreferenceType.MOUNTAINS_NEARBY, 1,
calculateWeight(1)),
                Preference(PreferenceType.TEMPERATURE, 0,
calculateWeight(2)),
                Preference(PreferenceType.PRECIPITATION, 1,
calculateWeight(3)),
                Preference(PreferenceType.SEA_NEARBY, 0,
calculateWeight(4)),
                Preference(PreferenceType.MINIMIZE_COSTS, 110,
calculateWeight(5)),
            )
        )

        val suggestions = getTripSuggestions(data, allPreferences)

        assert(suggestions.isNotEmpty())

        val firstSuggestions = suggestions.first().cities.map { it.name }
        assert(firstSuggestions.contains("Vienna"))
        assert(firstSuggestions.contains("Berlin"))
}
```

Отже, було проведене юніт тестування реалізованої програмної системи.

ВИСНОВКИ

Було розглянуто предметну галузь подорожей та побудови маршрутів. Було проаналізовано існуючі сервіси для створення планів подорожей та купівлі квитків та виявлено, що немає універсального сервісу, який би виконував задачу побудови маршрутів та подорожей оптимізовано. Отже, проблема полягає в тому, що поїздки, особливо ті, що містять декілька міст у ланцюжку, необхідно підбирати та конструювати для користувачів на основі їх уподобань. Існуючі сервіси, пов’язані з подорожами квитків не мають можливості запропонувати такі детальні поїздки, вони дозволяють фільтрувати подорожі за місцем призначення та ціною, але цього недостатньо.

Було розглянуто існуючі методи багатокритеріальної оптимізації для ранжування опцій. До цих методів належать: лінійна адитивна згортка, мультиплікативна згортка, максимінна згортка. Також було розглянуто методи комбінаторики, а також методи вирішення задачі комівояжера. Після цього було поставлено завдання цієї роботи.

Було розроблено вдосконалений метод, який базується на таких складових: лінійна адитивна згортка з нормуючими множниками та ваговими коефіцієнтами, перестановки та сполучення з комбінаторики, задача комівояжера.

Після розробки методу було проведено експеримент.

Було спроектовано виконання алгоритму та описано основні модулі, що до цього відносяться. Обрано саме таку архітектуру для забезпечення гнучкості при модифікації та подальшому вдосконаленні системи. Архітектуру послідовності взаємодії було зображено у вигляді діаграми. Система матиме архітектуру з відкритими інтерфейсами, що надає змогу розширювати її іншим розробникам та забезпечить підтримуваність. Також було спроектовано архітектуру проекту мобільного застосунку.

Було розроблено програмну реалізацію алгоритму та мобільний застосунок, продемонстровано основні ділянки коду алгоритму та архітектуру проекту. Також

було представлено роботу програмного застосунку: введення побажань, вибір пріоритетів та отримання пропозицій щодо подорожі. А також було проведено мануальне тестування та автоматизоване юніт тестування.

За результатами атестаційної роботи опубліковано тези доповідей на двадцятій міжнародній науково-технічній конференції «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління», (див. додаток Е). Інші матеріали були представлені на конференції SoftEngine 2022 (див. додаток Ж), а також було подано матеріали на конференцію IntelITSIS-2022 (див. додаток И).

За результатами роботи було розроблено презентацію, яку наведено в додатку Л.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Hammond C. The Art of Rest. How to Find Respite in the Modern Age – UK, Canongate Books, 2019.

2. Mazurova O., Samantsov O., Topchii O., Shirokopetleva M. A Study of Optimization Models for Creation of Artificial Intelligence for the Computer Game in the Tower Defense Genre//2020 IEEE International Conference on Problems of Infocommunications. Science and Technology (PIC S&T), 2020, pp. 491-496, doi: 10.1109/PICST51311.2020.9468057.

3. Ногин В. Д. Лінійна згортка критеріїв у многокритеріальний оптимізації//Матеріали конференції Штучний інтелект та прийняття рішень – 2014.

4. Черноруцький І. Г. Методи оптимизації. Комп'ютерні технології – 2011 – 384 с.

5. Бондаренко М. Ф. Комп'ютерна дискретна математика: підручник / Бондаренко М. Ф., Білоус Н. В., Руткас А. Г. –Харків, «Компания СМИТ», 2004 - 485с.

6. Мудров В. І. Задача про комівояжера – 2013 – 64 с.

7. Куліков А. Алгоритми для задачі комівояжера. 2012.

8. Emelichev V., Pashkevich, A. On the linear convolution of criteria in vector discrete optimization//Diskretnyĭ Analiz i Issledovanie Operatsiĭ. Seriya 2 – 2020.

9. Kryvoruchko M., Shirokopetleva M., Vechur O. The system for trip planning//Дванадцята міжнародна науково-технічна конференція «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління»

10. Osis J., Donins, U. Topological UML Modeling: An Improved Approach for Domain Modeling and Software Development – 2017.

11. Vechur O., Kryvoruchko M., Shirokopetleva M. Research of Methods of Planning Tourist Trips//IntelITSIS-2022

12. Juszczak, P., Tax D., Dui R. Feature scaling in support vector data descriptions – 2002.

13. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language – 2003.

14. Kryvoruchko M., Shirokopetleva M., Vechur O. Trip planning software system design/ SoftEngine 2022

15. Using Kotlin for Android Development//Kotlin – URL: https://kotlinlang.org/docs/android-overview.html (дата звернення: 19.04.2020).

16. A combinatorics library for Kotlin//GitHub – URL: https://github.com/shiguruikai/combinatoricskt (дата звернення: 22.04.2020).

17. Guide to app architecture//Developers – URL: https://developer.android.com/topic/architecture (дата звернення: 24.04.2020).

18. Martin R. Clean Architecture: A Craftsman's Guide to Software Structure and Design – Prentice Hall, 2017. – 432 p.

19. Seemann M. Dependency Injection Principles, Practices, and Patterns – 2019 – 562 p.