НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НН Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

До захисту допущено
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____2023р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 «Системний аналіз»
на тему: «Середовище розв'язання задач сценарного аналізу на основі модифікованого методу морфологічного аналізу»

Виконав:
студент IV курсу, групи КА-92
Демчишин Остап-Тадей Назарович _____

Керівник:
старший викладач кафедри ММСА,
к.т.н., Савченко І.О. _____

Консультант з економічного розділу:
доцент кафедри ТПЕ, к.е.н., доц.
Рощина Н.В.

_____

Рецензент
старший викладач кафедри ШІ,
к.т.н., Шаповал Н.В.

_____

Консультант з нормоконтролю:
к.ф.-м.н., Статкевич В.М.

_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
НН Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 124 «Системний аналіз»
Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____2023р.

ЗАВДАННЯ

на дипломну роботу студента

Демчишина Остапа-Тадея Назаровича

1. Тема роботи «Середовище розв’язання задач сценарного аналізу на основі модифікованого методу морфологічного аналізу», керівник роботи Савченко І.О., к.т.н., затверджені наказом по університету від «___»_____ 2023 р. №_____

2. Термін подання студентом роботи __.06.2023

3. Вихідні дані до роботи: І. О. Савченко. Методологічне і математичне забезпечення розв’язання задач передбачення на основі модифікованого методу морфологічного аналізу

4. Зміст роботи: теоретичні відомості, програмна реалізація, розробка додатку, функціонально-вартісний аналіз програмного продукту

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): актуальність, морфологічний аналіз, структура додатку,

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В., доцент		

7. Дата видачі завдання:_____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Вибір теми і постановка дослідження	23.04.2023	Виконано
2	Аналіз актуальності задачі дослідження	28.04.2023	Виконано
3	Збір та аналіз літератури	05.05.2023	Виконано
4	Формулювання задачі дослідження	12.05.2023	Виконано
5	Розробка програмного продукту	16.05.2023	Виконано
6	Аналіз та впорядкування результатів	27.05.2023	Виконано
7	Оформлення пояснювальної записки	28.05.2023	Виконано
8	Оформлення презентації для демонстрації	29.05.2023	Виконано

Студент Остап-Тадей ДЕМЧИШИН

Керівник Ілля САВЧЕНКО

РЕФЕРАТ

Дипломна робота: 118 с., 18 рис., 7 табл., 2 додатки, 17 джерел.

Ключові слова: МОРФОЛОГІЧНИЙ АНАЛІЗ, ЗАДАЧА СЦЕНАРНОГО АНАЛІЗУ, ДВОЕТАПНИЙ МОРФОЛОГІЧНИЙ АНАЛІЗ.

Об’єкт дослідження: задачі сценарного аналізу на основі двоетапного метода морфологічного аналізу.

Мета дослідження: створити середовище розв'язання задач сценарного аналізу для розв'язку двоетапним методом морфологічного аналізу

Використані моделі: у програмній реалізації було використано модель модифікованого методу морфологічного аналізу.

Отриманні результати: створений веб-додаток який приймає вхідні дані задачі сценарного аналізу та виконуючи необхідні обчислення виводить результати кожного з етапів дослідження.

Програмний продукт було розроблено, використовуючи мову програмування JavaScript.

ABSTRACT

Thesis: 118 pages, 18 figures, 7 tables, 2 appendices, 17 references.

Keywords: MORPHOLOGICAL ANALYSIS, SCENARIO ANALYSIS PROBLEM, TWO-STAGE MORPHOLOGICAL ANALYSIS.

The object of the study: tasks of scenario analysis based on the two-stage method of morphological analysis.

The purpose of the research: to create an environment for solving scenario analysis problems for solving by the two-stage method of morphological analysis

Models used: the model of the modified method of morphological analysis was used in the program implementation.

Obtained results: a web application was created that accepts the input data of the task of scenario analysis and, performing the necessary calculations, outputs the results of each of the research stages.

The software product was developed using the JavaScript programming language.

ЗМІСТ

ВСТУП

Існують задачі у вирішенні яких виникають певні неточності або ж невизначеності об’єктів що розглядаються, також дані можуть бути неповні, в зв’язку з цим виникає багато інших способів реалізації. Модифікований метод морфологічного аналізу застосовується для якісного аналізу таких задач. Він дозволяє описати досліджуваний об’єкт, проаналізувати, вивчити та здійснити рішення, не зважаючи на невизначеності.

При проектуванні задачі вводиться система параметрів та альтернатив. При використанні методу відбувається моделювання все-можливих комбінацій альтернатив різних параметрів. Це дозволяє отримувати інноваційні системи що комбінують у собі альтернативи які раніше в реальному житті не були реалізовані та можуть принести суттєву цінність як досліднику так і людству загалом.

Проте, інноватівність цих систем є відносною, оскільки алгоритм працює лише із альтернативами які були задані у систему, відповідно аналізує та працює виняткова з ними, не беручи до уваги нічого ззовні, отже нові створені системи можуть виступати у ролі новітні. Виходить що самі параметри при цьому, або ж альтернативи будуть обмежені дослідником, та процедура не передбачає отримання певних нових значень чи уведених даних. Тому, інструмент залишається хорошим у сфері досліджень та аналітики, але відчуває певні труднощі з інноватівністю.

Виходячи з міркувань вище - двофакторний метод морфологічного аналізу може виступити хорошим інструментом якогось іншого методу, який використовуватиме його за призначенням, та дозволить уводити нові дані ззовні які суттєво впливатимуть на дослідження та можуть привести до отримання чого справді раніше невідомого.

Двофакторний метод морфологічного аналізу достатньо лаконічний та зручний у використанні. Проте виконання обчислень та аналіз, а у випадку

великої кількості параметрів, альтернатив та значень може бути вельми трудомістким. Враховуючи що щось може змінитися, або ж потрібно буде вносити корективи, процес взагалі може відбуватися надто довго. Тому виконувати його вручну може бути недоцільно зовсім. Натомість володіючи інтерфейсом з можливістю динамічних змін даних та значень, який би виконував обчислення машинально та моментально виводи отримані значення у дослідженнях суттєво зможе суттєво спростити виконання аналізу складних систем.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Морфологічні таблиці

Достатньо важливим етапом, що здійснюється аналітиком є побудова морфологічних таблиць. Це забезпечує зв’язок з реальним світом.

Морфологічна таблиця може здійснювати кілька варіантів опису, відповідно до сутностей з якими вона взаємодіє:

1. Опис об’єкта – розглядається матеріальний або ні об’єкт чи система. Тут невизначеність розглядається відносно об’єкту, через фактори які розглянемо нижче.
2. Опис стану – розглядається стан системи чи об’єкту. Тут невизначеність розглядається в рамках того що відбувається чи відбуватиметься в об’єкті дослідження, такий об’єкт вважається відомим.
3. Опис дії – розглядається деяка подія або взаємодія між об’єктами. Тут невизначеність розглядається в рамках контексту події, характеристиках перебігу події. Зазвичай система у якій взаємодіють об’єкти відома[15].

1.2 Причини невизначеності об’єкта дослідження в МММА

Об’єкт морфологічного дослідження має містити значну кількість конфігурацій, тобто альтернативних варіантів реалізації, що може бути спричинено такими факторами:

1. Відсутність детальної інформації про конфігурацію об’єкта. Може бути викликано тим, що дослідник з певних причин не може отримати детальнішу інформацію або ж через те що об’єкт може розглядатися після деякого часу чи певних подій, які йому передують.
2. Необхідно розглянути різні конфігурації об’єкта у сукупності. допомогою морфологічну таблицю описується множина об’єктів, характеристики або складові яких можуть бути відомими, а невизначеність полягає у тому, яка саме конфігурація альтернатив параметрів об’єкта з’явиться при наступній реалізації.
3. Невизначеність виникає у момент вибору конфігурації об’єкту. Так як невизначеність власне полягає у прийнятті рішення щодо конфігурації, а не через певні невідомі фактори ззовні. Такий тип об’єктів є найважчим щодо автоматичного видобування морфологічної таблиці, оскільки генерація параметрів і альтернатив є надзвичайно складною творчою задачею, яка не може бути вирішена за допомогою аналізу великих обсягів даних[15].

1.3 Постановка задачі експертного оцінювання

Модифікований метод морфологічного аналізу ставить за мету визначити ймовірності альтернатив параметрів при реалізації об'єкта, заданого морфологічною таблицею. Для цього спочатку необхідно отримати початкові наближення для ймовірностей альтернатив характеристичних параметрів. В ідеалі це мають бути незалежні ймовірності, однак для реальних задач виконання цієї умови практично неможливе. Для отримання цих величин пропонується застосувати експертне оцінювання.[15]

Так оцінки надані експертами є достатньо наближеними, оскільки не враховують що параметри можуть бути залежними, то для отримання кінцевих значень ймовірностей необхідно врахувати взаємозв’язки та здійснити обчислення ймовірностей альтернатив параметрів

1.4 Двоетапний метод морфологічного аналізу

В процесі сценарного аналізу часто буває доцільним застосування двоетапної процедури МММА. При цьому на першому етапі здійснюється аналіз неконтрольованих факторів, так званих факторів “зовнішнього світу” для об'єкта, проблеми або явища, що розглядається. Другий етап дослідження полягає в синтезі рішень, які найбільш ефективно враховувати в умовах сукупності можливих реалізацій об'єкта, визначених на першому етапі.

Таким чином, будуються дві пов'язані морфологічні таблиці: морфологічну таблицю сценаріїв та морфологічну таблицю стратегій. Специфіка другого етапу МММА полягає в тому, що вибір альтернатив параметрів МТ стратегій залежить не від випадкових зовнішніх факторів, а

від особи, що приймає рішення, тому немає сенсу говорити про ймовірність вибору альтернатив. Тому на другому етапі для оцінки альтернатив і конфігурацій використовується величина очікуваної результативності, тобто вірогідності того, що вибір цієї альтернативи або конфігурації призведе до бажаних результатів.[15]

1.5 Висновки до розділу 1

У цьому розділі було розглянуто теоретичні відомості щодо методу морфологічного аналізу, який буде взято за основу створення додатку для його використання. Була поставлена задача експертного оцінювання та описано двоетапний метод морфологічного аналізу.

Також була приділена увага причинам невизначеності об’єктів дослідження у ММА, а також розглянуто питання морфологічних таблиць

РОЗДІЛ 2
ПРОГРАМНА РЕАЛІЗАЦІЯ

2.1 Структура

2.1.1 Основні модулі

Програмна реалізація передбачає створення:

1. Користувацького інтерфейсу, для зручної, простої та зрозумілої взаємодії користувачів, відображення уведених даних, результатів обчислень.
2. API – прошарку, який витягуватиме дані з інтерфейсу, зберігатиме їх у базі даних, зі змогою витягувати їх звідти та виконуватиме усі необхідні обчислення.
3. Бази даних, де зберігатимуться уведені користувачем дані, та задля їхньої подальшої обробки API – прошарком.

Планується надати можливість користувачу задавати характеристичні параметри, відповідно з експертними оцінками, а також матрицю взаємозв’язків альтернатив параметрів. Після виконання усіх обчислень, будуть виводитися відповідні результати першого та другого етапів. Планується зберігати вхідні дані, та результати у базі даних(див. рисунок 2.1).

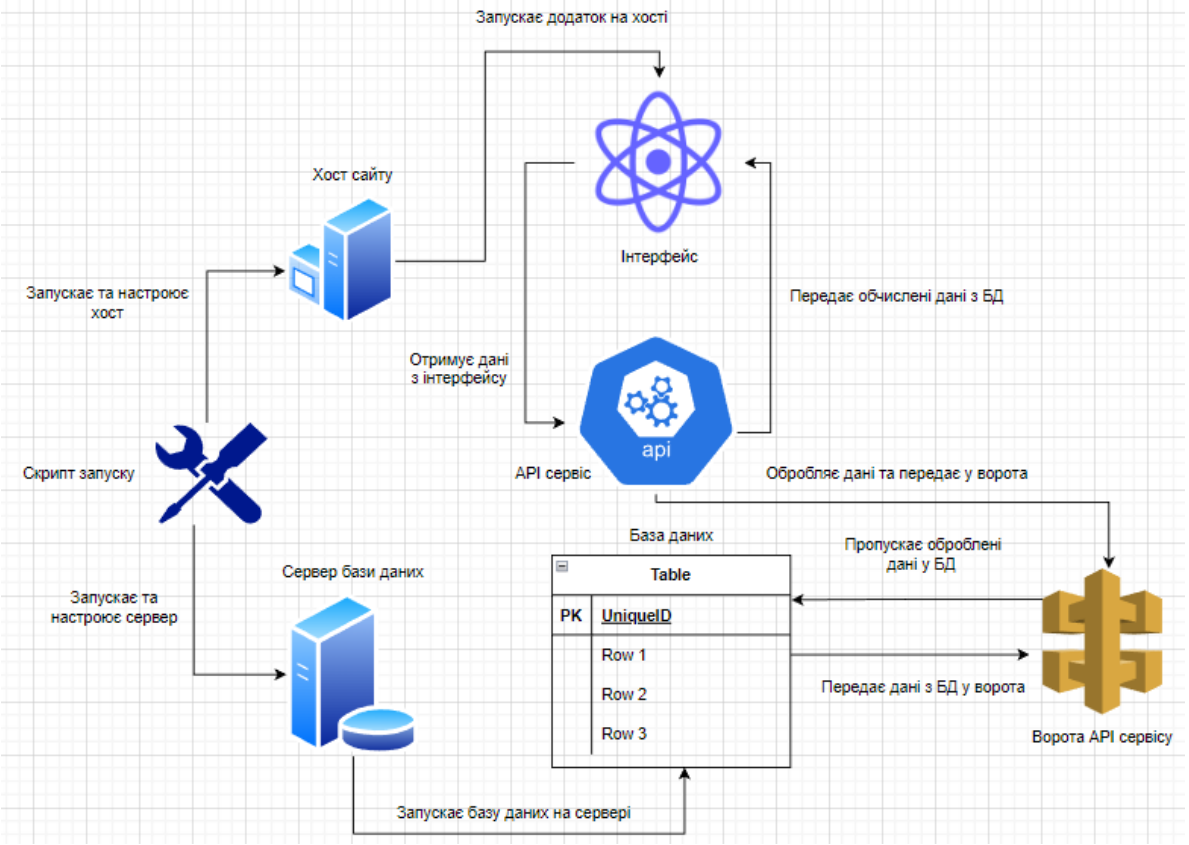


Рисунок 2.1 – Діаграма структури веб-додатку

2.1.2 Список основних елементів

1. Інструмент(скрипт)запуску додатку;
2. Локальний хостинг додатку;
3. Сервер розгортання БД;
4. Інтерфейс додатку;
5. API сервіс;
6. Ворота АПІ сервісу;
7. БД.

2.1.3 Детальний опис взаємодії модулів

Почнемо розгляд додатку з його запуску. Для його успішного виконання необхідно правильно налаштувати скрипт запуску. Так як додаток складається з двох частин, необхідно здійснити це і для бази-даних та API-сервісу, і для інтерфейсу додатку:

1. Для запуску інтерфейсу будемо використовувати вбудоване середовище запуску ядра Node node.js.[3]. Скрипт виглядатиме наступним чином: start: node app.js Для його запуску використаємо вбудований у Node npm пакет: npm run start [12]. Додамо синтаксичного цукру для інтуїтивної зрозумілості у вигляді скрипта клієнта: client: npm run start.[11]
2. Для запуску серверу будемо використовувати інструмент запуску nodemon. Скрипт виглядатиме наступним чином: server: nodemon app.js [7]. Для його запуску використаємо вбудований у Node npm пакет: npm run server.
3. Аби обидві частини можна було запустити одночасно і однією командою(як для повноцінного додатку) використаємо бібліотеку conqurently [10]. Скрипт виглядатиме наступним чином: dev: conqurently \”npm run server\” \”npm run client”\. Для його запуску використаємо вбудований у Node npm пакет: npm run dev[9].

На цьому кроці у нас запуститься сервер для БД та API сервісу, а також інтерфейс додатку, після чого він буде готовий до використання.

Розглянемо макет інтерфейсу(див. рисунок 2.2, рисунок 2.3 та рисунок 2.4)

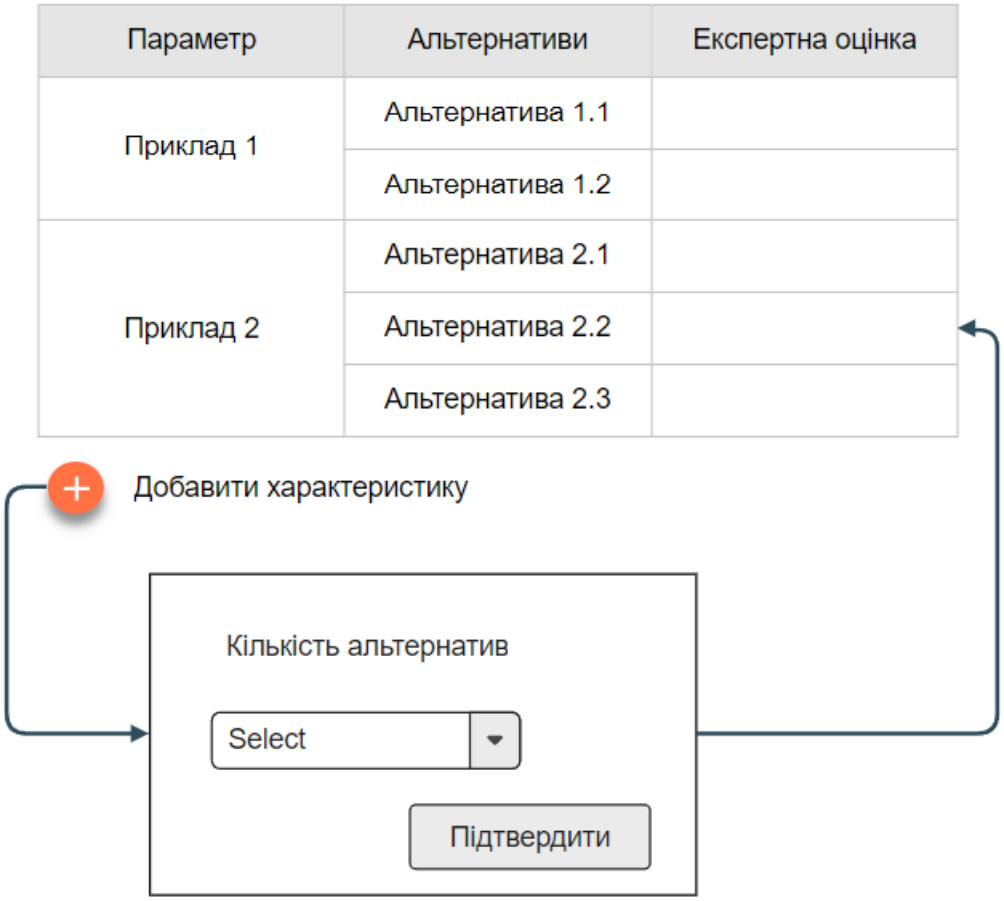


Рисунок 2.2 - Додавання характеристик та альтернатив

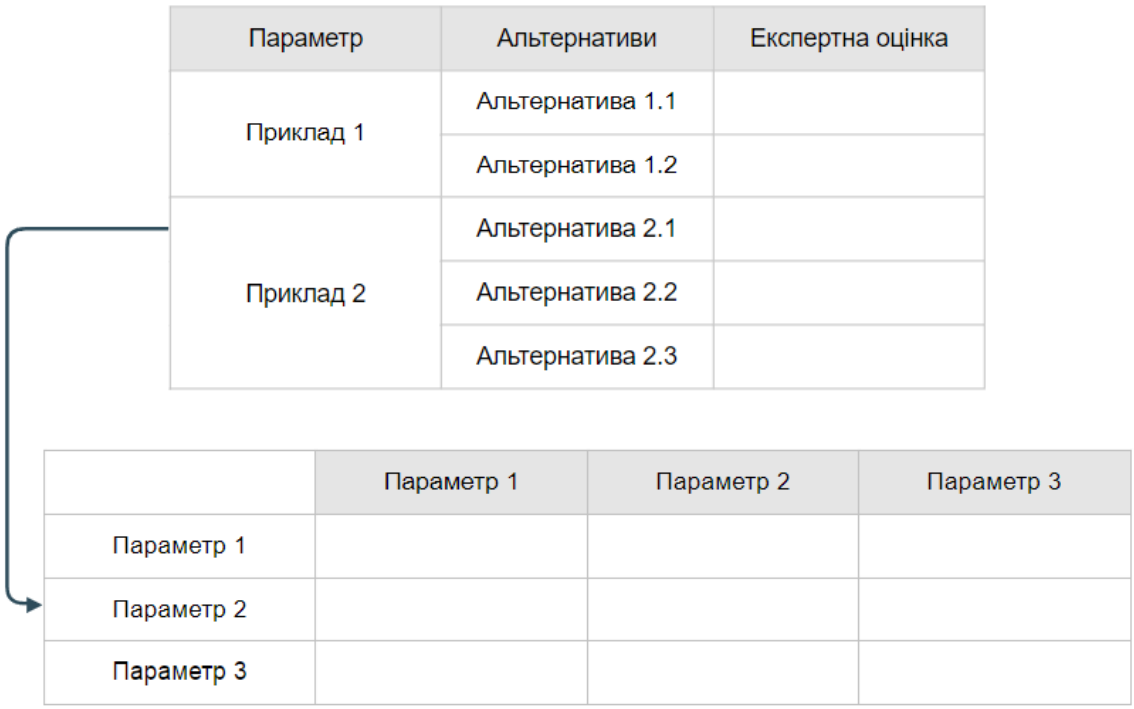


Рисунок 2.3 – Матриця взаємозв’язків

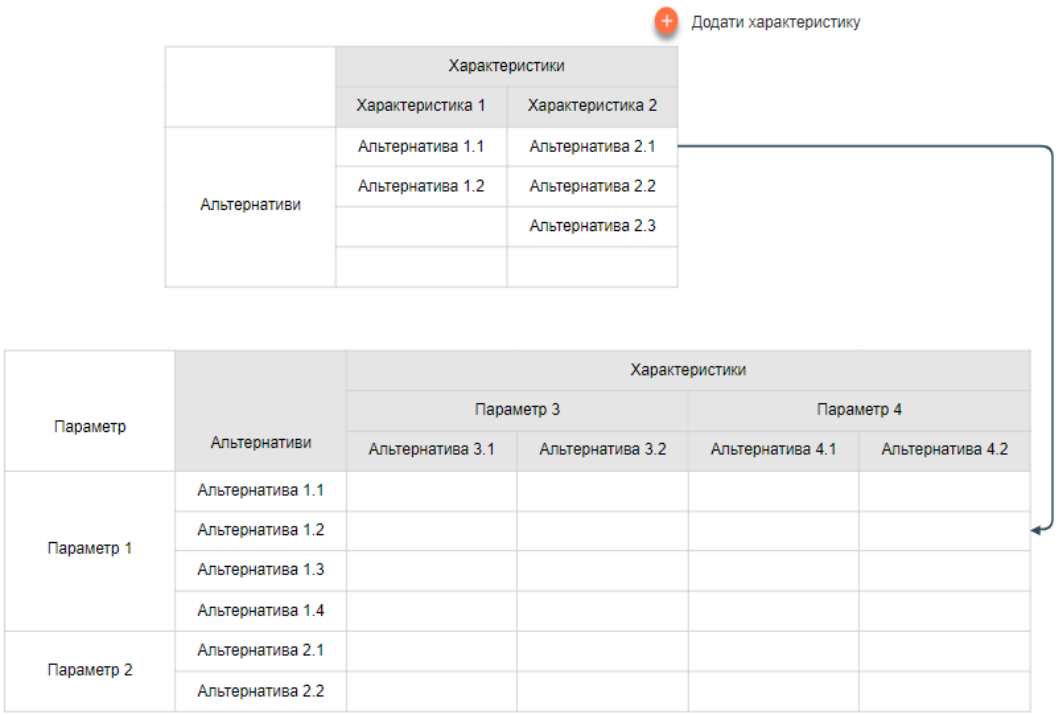


Рисунок 2.4 - Морфологічна таблиця другого етапу

Виведення результатів відбуватиметься у аналогічному форматі. Користувачу надається можливість уведення характеристик та альтернатив, а також відповідні експертні оцінки та матрицю взаємозв'язків характеристик. Після збереження уведеного інформація потрапляє у API – сервіс, обробляється ним та зберігається у БД. Основною сутністю додатку буде характеристика, яка у БД матиме набір значень взаємозв'язків з іншими характеристиками, та міститиме інформацію про всі обчислення які мають із нею залежність. Також для збереження проміжних результатів обчислень, значень великих таблиць використовуватимуться додаткові сутності, які будуть залежні із сутностями характеристик. Для початку здійснення обрахунків буде виведена додаткова кнопка, після чого запуститься процедура, яка виконуватиметься на стороні АПІ-сервісу, та відразу зберігатиме проміжні результати у БД. У цей час на екрані користувача буде зображений СПІНЕР, який позначить прохання зачекати на обчислення. Після закінчення процедури, ми отримаємо результат роботи, у вигляді очікуваної результативності альтернатив. Надалі можна буде отримати детальнішу

інформацію про обчислення у вигляді проміжних таблиць, які відтворюватимуться на прохання користувача.

Для введення даних будуть використовуватися вбудовані input інструменти React, та React таблиці по типу mui пакету, а для виведення аналогічно. [6]

АПІ–сервіс виконуватиме усі обчислення на своєму боці, причому використовуватиме методику middleware згадану нижче, а також буде взаємодіяти з БД за принципом back service–data base–back service.

Формат та схема збереження даних у БД розглядатиметься у ході виконання частини збереження даних, передбачити це достатньо складно, тому варто відштовхуватися від обставин які виникатимуть у процесі розробки

Під час взаємодії користувача з додатком для виведення даних, при натиску на відповідні кнопки, буде створюватися АПІ-запит який витягуватиме відповідні дані з БД, та передаватиме їх до інтерфейсу, після чого там відображатиметься за допомогою вищезгаданих інструментів роботи з таблицями. Важливо, що при виведенні одних даних, для отримання будуть закриватися попередні, це буде здійснено для оптимізації роботи інтерфейсу, інакше грозить довге завантаження при великих об’ємах даних.

Усі вище згадані модулі будуть запускатися локально на машині розробника, на порті вказаному у пакеті опису проєкту додатку.

2.2 Інтерфейс

2.2.1 Основи React

Даний підрозділ використовує опис, наведений в [1].

Розглянемо частину програмної реалізації користувацького інтерфейсу: основні інструменти та механізми для створення інтерфейсу використовуючи бібліотеку React.

Задля створення зручної фронтальної компоненти було вирішено використовувати бібліотеку для розробки користувацького інтерфейсу React, засновану на мові програмування JavaScript. Це популярний сучасний інструмент і його основна задача – забезпечити відтворення вмісту веб-сторінок.

2.2.2 SPA

Даний підрозділ використовує опис, наведений в [13].

React може використовуватися для розробки додатків однієї сторінки – SPA(single page application) чи мобільних додатків. Ми ж будемо використовувати для першого. Бібліотека дає змогу швидко розробляти та масштабувати додатки. Вона значно спрощує роботу, на відміну від звичайних мов розмітки(HTML), за рахунок розбиття на багато різних компонент, зазвичай невеликих розмірів, які інтуїтивно зрозуміло виглядають.

Додатки однієї сторінки – сайти або ж додаток що використовують як оболонку один HTML документ для усіх сторінок. А рухливі елементи сторінки, чи ті що призначені для користувача динамічно завантажуються додатково до документу: HTML, CSS та JavaScript використовуючи AJAX.

2.2.3 AJAX

Даний підрозділ використовує опис, наведений в [8].

AJAX – asyncronous javascript and XML – модель розробки інтерфейсів з можливістю взаємодії користувача який полягає в обміні інформацією браузера з веб-сервером у фоновому режимі. Тому немає необхідності перезавантажувати такий сайт повністю, коли змінюється один або кілька фрагментів. Цей принцип буде основоположним у нашому веб-сервісі.

2.2.4 Функціональні компоненти

В рамках бібліотеки і далі елементи будемо називати КОМПОНЕНТАМИ. Компонент – частинка коду яка відображає певну фрагмент на сторінці. Щоправда у React ці компоненти є функціями, які повертають, власне, код-розмітку що відображає фрагмент веб-сторінки. Для формування сторінки викликаються усі необхідні функції певному порядку.

```
function ReactComponent() {
  return (
  )
}
```

Ця бібліотека використовує мову програмування JSX – візуально схожу на HTML, але суттєвою відмінністю є те, що він запускається середовищі JavaScript.

```
function ReactComponent() {
  return (
      <p> Some chapter </p>
  )
}
ReactDOM.render(<ReactComponent/>, placeForComponent)
```

Коли ми викликаємо функцію вона поверне фрагмент JSX – коду, після чого відмалює його у DOM – дереві нашого документу.

2.2.5 DOM – дерево

Даний підрозділ використовує опис, наведений в [8].

DOM(Document object model) – дерево – відображення HTML документу у вигляді дерева тегів мови HTML, які складають основу об’єктної моделі документа. Усі теги вважаються об’єктами. Усі з них доступні для отримання за допомогою мови програмування JavaScript – для подальшої взаємодії.

Приклад шаблону HTML документу:

```
<!DOCTYPE HTML>
<html>
<head>
  <title>About React</title>
</head>
<body>
  Detailed info
```

```
</body>
</html>
```

2.2.6 Вкладеність компонентів

Необхідною особливістю React є можливість вкладення компонент одна у одну.

```
function ReactContainer() {
  return (
    <div>
       <ReactComponent/>
    </div>
  )
}
ReactDOM.render(<ReactContainer/>, placeForComponent)
```

Особливістю є направлена в одну сторону передача даних – властивості передаються від батьківських до дочірніх. Компоненти отримують властивості як множину незмінних значень, тому компонент не може напряму змінювати властивості, лише за допомогою callback – функцій. Даний механізм називають: «властивості вниз – події нагору»(про стани та події далі).

2.2.7 Класові компоненти

Також важливим є підтримка бібліотекою класової структури. Якщо раніше ми розглядали компоненти як функції, то зараз, бачимо, що можна використовувати класові компоненти.

```
class ClassReactComponent extends React.Component {
 render() {
  return (
   <div>
    <ReactComponent />
   </div>
  );
 }
}
ReactDOM.render(<ClassReactComponent />, placeForComponent);
```

Варто добавити, що випадку класових компонент необхідна наявність render() – функції для коректної обробки та відмалювання елементів.

Звертатися до класових компонент можемо аналогічно до функціональних.

2.2.8 JSX

Даний підрозділ використовує опис, наведений в [8].

JSX – код особливий тим, що у нього можна вставляти змінні, фрагменти коду та навіть запускати всередині функції JavaScript.

```
class ReactContainer extends React.Component {
 render() {
  const stringToShow = 'String to show!';
  return (
   <div>
```

```
        <p>{ stringToShow }</p>
        <ReactComponent />
      </div>
    );
  }
}
```

Запуск функції:

```
class ReactContainer extends React.Component {
  render() {
    const multiplyFunc = (num1, num2) => {
      return num1 * num2;
    };
    return (
      <div>
        <h1>The res is: { multiplyFunc(5, 6) }</h1>
        <ReactComponent />
      </div>
    );
  }
}
```

Варто наголосити що class – ключове слово мови JavaScript, тому JSX не може використовувати його для присвоєння класу певному елементу, для цього є аналог JSX – className.

```
function InputToShow() {
    return <input className="common input" />;
  }
```

Якщо нас цікавлять компоненти зі станом, тоді перевагу варто надавати класовим, якщо ж ні – перевага на стороні функціональних.

2.2.9 Стани компонентів

Компоненти які базуються на класах можуть зберігати інформацію про поточну ситуацію. Ця інформація називається станом(state) компоненту, і зберігається у JavaScript об’єкті.

```
class ReactClassContainer extends React.Component {
 constructor(props) {
  super(props);
  this.state = { isDefined: false };
 }

 render() {
  return (
   <div>
    <InputToShow />
   </div>
  );
 }
}
```

Метод constructor, який відповідає за однойменне конструювання об’єкту завжди викликає super метод для конструювання спершу батьківського елементу – це обов’язкова особливість для збереження принципу наслідування класів та передачі методів та полів у спадок.

Стан – інструмент що забезпечує оновлення наповнення інтерфейсу в залежності від подій. Користувач може взаємодіяти з компонентом, і щоб реагувати на ці події існують функції – обробники подій.

```
class ReactClassContainer extends React.Component {
 constructor(props) {
  super(props);
  this.state = { isDefined: false };
 }
 handleInput(event) {
  // Реакції на ввід даних
 };
 render() {
  let status = this.state.isDefined
  ? 'Entered'
  : 'Not entered';
  return (
   <div>
    <input onChange={this.handleInput.bind(this)}>{ status }</h1>
    <InputToShow />
   </div>
  );
 }
```

Коли користувач тисне на кнопку компонент викликає функцію: функція отримує об’єкт події в якості аргументу і може його використовувати

У момент взаємодії, після виклику функції компонент самостійно викликає функцію render() та перемалює себе ж, за потреби може змінити стан за допомогою функції setState(). У цієї функції є особливість, оскільки при її запуску зміна стану не відбувається миттєво, перед цим React перевіряє чи будуть ще якісь зміни станів і тільки тоді вносить їх.

```
handleInput() {
```

```
  if (this.state.isDefined) {
    this.setState({ isDefined: false });
  } else {
    this.setState({ isDefined: true });
  }
};
```

2.2.10 React хуки

Даний підрозділ використовує опис, наведений в [8].

Коли суспільство вперше познайомилося з функціональними компонентами, вони служили лише для того, щоб виводити передану інформацію. Тому не володіли поняттям стану, чи методів життєвого циклу. Компоненти були простими наскільки це можливо. Тому, часто виникали, ситуації, коли компоненті, написаній у функціональному стилі, необхідно було володіти станом або ж методом життєвого циклу, а такої можливості фізично не було. Доводилося переписувати їх у класові компоненти, а це достатньо трудоємно та часозатратно.

Такий стан справ підштовхнув команду React до створення хуків, що дозволяють розширити можливості функціональних компонентів або нівелювати деякі проблеми, які можуть виникати через їхню специфіку. Хуки виявилися настільки зручними, що стали основою React-розробки.

Почнемо з найпростішого і найважливішого хука - useState. З назви стає зрозуміло, що він пов'язаний зі станом компонента. Саме завдяки ньому у функціональних компонентів з'явилося поняття стану. Ось приклад як його можна використати:

```
const App = () => {
```

```
  const [value, valueChanger] = useState(0);

  return (
    <div>
      {value}
      <button onClick={() => valueChanger(value + 1)}>
        Додати одиницю
      </button>
    </div>
  );
};
```

Логіка полягає у тому що при натисканні кнопки змінюється стан змінної value, додаючи до неї 1. За процес зміни значення змінної відповідає функція valueChanger, яку повертає як параметр хук useState. Як початкове значення змінній value присвоюється значення параметру переданого у хук. По суті хук повертає масив двох параметрів: стан і метод який буде його змінювати. За зв'язок між натиском кнопки та зміною стану відповідає обробник подій onClick(), у який передається параметром стрілочна функція з методом зміни стану.

Інші властивості як із звичайним станом компонента. Основна відмінність полягає в тому, що у класовому компоненті ми можемо створити лише один загальний стан компонента, а у функціональному – кілька, і вони будуть незалежні один від одного, але кожне зі станів при його зміні викликатиме функцію малювання компонента повністю.

Щоб передати певні дані у компоненту, ми знову ж таки можемо використовувати props та безпосередню передачу від батьківського до дочірнього елементу властивостей. Але є альтернативний спосіб – context.

Контекст дозволяє передавати дані від батьківського компонента до дочірнього, оминаючи проміжні.

Розглянемо три компоненти. Кожен з компонентів буде вкладений один в одного. Наше завдання – передача даних із компонента Parent до компоненту Child, минаючи Middle, оскільки для проміжного елементу дані що передаються відношення не мають.

```
import {createContext, useContext} from "react";

const MyContext = createContext("Random value");

const Parent = () => {
  return (
    <MyContext.Provider value="Parent element">
      <Middle />
    </MyContext.Provider>
  );
};

const Middle = () => {
  return <Child />;
};

const Child = () => {
  const context = useContext(MyContext);
  return `Here is child component. The data from external level: "${context}"`;
};
```

Щоб використовувати контекст, ми створюємо об'єкт MyContext, викликаючи метод createContext. У компоненті Parent обгортаємо компонент Middle компонентом MyContext.Provider. Тим самим оголошуємо, що всі вкладені в нього компоненти зможуть отримати доступ до даних, які ми передаємо у параметрі value.

Причому вони будуть доступні лише у тих компонентах, у яких ми вкажемо. Для цього ми повинні використовувати хук useContext, а як аргумент у нього буде об'єкт MyContext. Хук useContext поверне нам дані, передані в параметр value MyContext.Provider, які ми помістимо в змінну context. При створенні контексту ми передали рядок. Його значення потрапить у змінну context у випадку, якщо не буде використовуватися обгортка MyContext.Provider, цей механізм попередить виникнення помилки, у випадку коли дані для value поля будуть недоступними, тоді буде передано цей рядок у вигляді аргументу.

Завдяки хуку useContext можна використовувати context у функціональних компонентах, і дані потраплятимуть лише у ті компоненти, у яких вони потрібні.

Ці хуки оживляють компоненти, а взагалі кажучи, надають методи життєвого циклу.

Методи життєвого циклу передбачені для того щоб здійснювати якісь операції на різних стадіях життя компоненти. Для цього у нас є два хуки - useEffect і useLayoutEffect. Загалом вони схожі між собою, за винятком невеликої різниці у малюванні. Щодо useLayoutEffect React не запускає малювання побудованого DOM дерева до того часу, поки не відпрацює useLayoutEffect. В свою чергу для useEffect React відразу запускає малювання побудованого DOM, не чекаючи запуску useEffect.

За допомогою цих двох хуків у функціональних компонентах можна змоделювати роботу трьох методів життєвого циклу – componentDidMount, componentDidUpdate, componentWillUnmount. Більш точно їхню роботу показує useLayoutEffect, тому що в класових компонентах малювання DOM-дерева не запускається до тих пір, поки не відпрацює метод componentDidMount.

Оскільки ці два хуки мають один і той же мехінізм, продемонструємо його на популярнішому - useEffect, а для іншого все буде аналогічно useEffect приймає два аргументи:

1. Callback. Сюди можна передавати будь логіку яку необхідно виконати. Наприклад, можна робити запити на сервер або задати обробник подій;
2. Масив аргументів. При зміні значення будь-якого елементу масиву запускатиметься callback функція. Саме завдяки цьому аргументу ми можемо імітувати методи життєвого циклу.

Розглянемо імітацію componentDidMount. Хук useEffect запускається не тільки при зміні елементів масиву з другого аргументу, але й після монтування компонента. Фактично componentDidMount запускається на тій самій стадії. Якщо ми вкажемо порожній масив як другий аргумент, callback запуститься на стадії монтування компонента. А оскільки жодних залежностей для хука всередині масиву ми не поставили, то аргумент callback не запускатиметься більше.

Імітація componentDidUpdate також можлива. Але у випадку з хуками є кращий спосіб. Оскільки ми можемо вказати в масиві лише ті залежності, які нам потрібні, ми маємо більш гнучкий аналог методу componentDidUpdate, який запускається при зміні необхідних параметрів. Якщо потрібно зробити аналогію componentDidUpdate, як залежність можна вказати всі параметри і стани.

Перейдемо до імітації componentWillUnmount. Для цього просто повертаємося з useEffect - callback. Практично повертається callback – це аналог componentWillUnmount, який часто застосовується для відв'язування обробників подій документа. У випадку з функціональним компонентом ми будемо відв'язувати їх всередині callback, що повертається.

Використовується коли необхідно звернутися до об’єкту DOM-дерева напряму.

```
const App = () => {
  const reference = useRef();

  useEffect(() => {
```

```
  console.info(reference.current);
 }, []);

 return <div ref={reference} />;
};
```

Ми створюємо об'єкт reference і вказуємо його як елемент, який позначає DOM-об'єкт, до якого ми хочемо звернутися, а також прописуємо цей об'єкт як параметр. Далі ми можемо взаємодіяти з Dom-об'єктом безпосередньо, так ніби ми знайшли його за допомогою селектора. З цією метою використовуєьбся властивість current у об'єкта ref.

Крім цього, можна застосувати useRef, якщо нам потрібно буде запам'ятати дані в компоненті. Але взагалі не рекомендується викликати малювання в разі зміни стану компонента.

Хук який відтворює функціонал бібліотеки Redux, що в свою чергу спрощує та уніфікує використання інструменту без застосування додаткових бібліотек.[16]

```
import {useReducer} from "react";

const initialState = {inProgress: false};

function reducer(state, action) {
 switch (action.type) {
  case "requested":
   return {
    ...state,
    inProgress: state.inProgress ?? !state.inProgress,
   };
  case "denied":
   return {
    ...state,
```

```
        inProgress: state.inProgress ? !state.inProgress : state.inProgress,
      };
    default:
      throw new Error();
  }
}

const App = () => {
  const [state, dispatch] = useReducer(reducer, initialState);

  return (
    <>
      {state.inProgress}
      <button onClick={() => dispatch({type: "requested"})}>-</button>
      <button onClick={() => dispatch({type: "denied"})}>+</button>
    </>
  );
};
```

З вище написаних рядків коду стає зрозуміло, що це зручний інструмент для керування станом даних і користувацьким інтерфейсом у додатках.

Використовується для оптимізації обчислень. Дозволяє виконувати одні і ті ж обчислення скільки необхідно разів. Розглянемо наступне.

```
const Component = ({a, b}) => {
  const aSquare = a * a;

  return (
    <div>
      <div>{ aSquare }</div>
```

```
    <div>B: {b}</div>
   </div>
  );
};
```

У цій ситуації компонент перемальовується у тому випадку, коли змінюється один із параметрів – a або b. Припустимо, що у нас багато разів змінюється параметр b, при цьому параметр a залишається незмінним. У такому разі ми багато разів обчислюємо той самий вираз, який поміщаємо в змінну aSquare. Але тоді перший параметр a у такому разі залишається без змін. Виходить, ми зайвий раз навантажуємо наш комп'ютер обчисленнями одного і того ж самого. І хоча в даному випадку операція виразу не надто трудоємна, в інших ситуаціях можливе набагато більше зайве навантаження. Уникнути надлишкових обчислень нам допомагає хук useMemo. Перетворимо наш приклад, до наступного вигляду:

```
const Component = ({a, b}) => {
 const aSquare = useMemo(() => a * a, [a]);

 return (
  <div>
   <div>{ aSquare }</div>
   <div>B: {b}</div>
  </div>
 );
};
```

Тут все залишилося, як і раніше, за винятком того, що ми обгорнули наш вираз у хук useMemo, в який передали callback і масив залежностей. По суті вони працюють так само, як і в useEffect: як тільки змінюється якась залежність з масиву, запускається callback, який розраховує інше значення. Якщо жодна залежність не змінилася, то при малюванні змінної буде підставлено попереднє обчислене значення.

Використовується задля ще більшої оптимізації. Нехай ми створюємо всередині компонента функцію та передаємо її в дочірній компонент. Ця практики часто зустрічається, коли нам потрібно з дочірнього компонента змінити щось у батьківському. Глянемо на приклад.

```
const Randomizer = memo(({changer}) => {
 return (
  <div>
   <button onClick={changer}>Random!</button>
  </div>
 );
});

const App = () => {
 const [value, randomChange] = useState(Math.random());

 const changer = () => randomChange(Math.random());

 return (
  <div>
   {value}
   < Randomizer changer={ changer } />
  </div>
 );
};
```

У цьому прикладі представлені два компоненти, один із них – Randomizer, який відповідає елементу з контейнером. У ньому лише одна кнопка, яка змінює стан батьківського компонента. Як параметр в нього передано метод changer, який у собі містить виклик методу randomChange, він і оновлює стан. Для простоти змінимо значення стану, просто помістивши туди довільне число. Ми спеціально обгорнули Randomizer у memo, щоб цей

компонент перемальовувався лише в тому випадку, якщо змінили його параметри. Однак у даному випадку у нас виникає проблема: при кожному малюванні компонента App ми будемо заново створювати метод changer, отже, у Randomizer відбуватимуться повторні малювання, але насправді нічого не змінюватиметься. І тут як параметр будуть передаватися різні реалізації однієї і тієї ж функції. Уникнути цього допоможе використання Callback.

```
const Randomizer = memo(({changer}) => {
  return (
    <div>
      <button onClick={ changer }>Random!</button>
    </div>
  );
});

const App = () => {
  const [value, randomChange] = useState(Math.random());

  const changer = useCallback(() => randomChange(Math.random()), []);

  return (
    <div>
      {value}
      < Randomizer changer={changer} />
    </div>
  );
};
```

Завдяки цьому хуку ми один раз створюємо метод і оновлюємо його лише тоді, коли змінюється якийсь із параметрів, які є другим параметром у хуку у вигляді масиву – за аналогією з масивами в інших хуках.

Це ті ж функції, які під капотом використовують якийсь із стандартних хуків. Єдина вимога, якої тут необхідно дотримуватися – відноситися до них, як до хуків. Тобто дотримуватись поведінки, яку ми використовуємо при роботі з хуками: не викликати їх усередині умовних конструкцій (if або switch) і всередині циклів (for…), а також не використовувати хуки всередині callback – функцій інших хуків.

Для того щоб було інтуїтивно зрозуміло що функція є хуком варто називати їх у форматі use«назва хука». Розглянемо приклад хука користувача:

```
const useExample = () => {
  useEffect(() => {
    console.info("Example");
  }, []);
};
```

Обмежень щодо створень таких конструкцій, окрім правил використання немає, тому, фактично, межею є границя необхідного функціоналу.

2.3 API – сервіс

Розглянемо частину програмної реалізації зв’язку інтерфейсу з базою даних, та обчислень: основні інструменти та механізми для створення API сервісу використовуючи фреймворк Express.

Express – гнучкий та мінімалістичний програмний каркас для розробки Node.js додатків. [14]

Для початку роботи необхідно створити екземпляр express додатку, запустити на обраному порті HTTP – сервер та добавляти маршрути для здійснення запитів на сервіс.

```
const app = express()
app.get('/', (req, res) => res.send('GET request was performed'))
app.listen(5000, () => console.log('App is listening on port 5000!'))
```

Враховуючи його надбудову над node.js, та доступні методи – даний фреймворк використовується для створення API сервісів з REST архітектурою.

2.3.1 REST – архітектура та HTTP

Даний підрозділ використовує опис, наведений в [8].

REpresentational State Transfer – один з найбільш популярних архітектурних підходів для створення API сервісів. Особливістю такої архітектури сервісів є те, що вони перевикористовують протокол HTTP, але з певними покращеннями.

Протокол HTTP(Hyper Text Transfer Protocol) є основоположним у мережі інтернет. При переході в браузері по URL – адресі: на сервер

відправляється запит, після чого сервер формує і надсилає відповідь. Власне формат запиту та відповіді відбувається за HTTP протоколом.

HTTP забезпечує базовий рівень створення веб-сервісів. Тому важливо розуміти основу HTTP. У нас є кілька базових абстракцій необхідних для розуміння.

Ресурс — ключова абстракція, де концентрується протокол HTTP. Ресурс - це все, що ми хочете показати назовні через наш додаток. У випадку, якщо ми пишемо додаток для управління завданнями, у нас будуть наступні екземпляри ресурсів:

1. Користувач.
2. Завдання.
3. Список завдань.

При розробці RESTful сервісів, необхідно зосередити свою увагу на ресурсах програми. Нам необхідно визначити який ресурс потрібно надати. Для цього потрібно використати URI – універсальний ідентифікатор:

1. Створити завдання: POST /tasks.
2. Видалити завдання: DELETE /tasks/1.
3. Отримати всі завдання: GET /tasks.
4. Отримати одне завдання: GET /tasks/1.

Важливо, що з REST необхідно розглядати додаток з точки зору ресурсів.

Для виконання операцій з цими ресурсами, потрібно використовувати ключові слова, визначені протоколом HTTP,

Загальний шаблон реалізації сервісу REST. Немає жодних обмежень на формат передачі даних. Зазвичай використовують JSON, також корисним але менш популярним є XML. Транспортування даних завжди відбувається за використання протоколу HTTP.

Визначення сервісу - немає стандарту, REST - гнучкий. Це може бути недоліком у деяких сценаріях, оскільки додаток який отримує дані може вимагати розуміння форматів запитів і відповідей.

Ресурси є основним елементом REST, також важливо, наскільки ефективно ми виконуємо операції з ресурсами, використовуючи HTTP. HTTP визначає наступну структуру запиту:

1. Рядок запиту (request line) – тип запиту.
2. Заголовки запиту (header fields) - характеризують тіло запиту, параметри передачі та інші відомості.
3. Тіло повідомлення (body) - необов'язкове поле з додатковою інформацією запиту.

HTTP визначає наступну структуру повідомлення у відповідь (response):

1. Рядок стану (status line), що включає код стану та повідомлення про причину.
2. Поля заголовка відповіді (header fields).
3. Додаткове тіло повідомлення (body).

Метод, що використовується в HTTP-запиті, вказує, яку дію ви хочете виконати із цим запитом. Наприклад:

1. GET: отримати інформацію про ресурс.
2. POST: створити новий ресурс.
3. PUT: оновити існуючий ресурс.
4. DELETE: Видалити ресурс.

Код статусу відповіді HTTP. Код стану завжди є у відповіді HTTP.
Наприклад:

- 200 – успіх,
- 403 — заборонено.

2.3.2 Express Middleware

Додаток Express - це по суті серія викликів функцій middleware. На перший погляд це звучить просто, але, взагалі кажучи, проміжне програмне забезпечення може бути дуже заплутаним.

Паттерн middleware є основним при створенні програм Express, тому необхідно добре розуміти, що таке middleware і як воно працює.

Шаблон Middleware. Express Middleware - це певний стиль функцій, які ми налаштовуємо для використання додатком. Вони можуть виконувати будь-який код, який необхідно, часто виконують роль обробників вхідних запитів, надсилання відповідей та обробку помилок.

Коли ми визначаємо маршрут Express, функція-обробник маршруту, яку ми вказуємо для цього маршруту, є функцією Middleware:

```
app.get("/task", function routeHandlerMiddleware(request, response, next)
{
   // execution
});
```

Middleware є гнучким інструментом. Ми можемо вказати Express запускати одну і ту ж функцію middleware для різних маршрутів, що дозволяє нам виконувати загальні дії для різних кінцевих точок API. Синтаксис Middleware.

```
function routeHandlerMiddleware(request, response, next) {
   // execution
});
```

Коли Express запускає функцію middleware, їй передаються три аргументи:

1. Об'єкт запиту Express (req/request) - це розширений екземпляр вбудованого в Node.js класу http.IncomingMessage.
2. Об'єкт відповіді Express (res/response) - це розширений екземпляр вбудованого в Node.js класу http.ServerResponse.

3. Функція Express next() - Після того як проміжна функція виконає свої завдання, вона повинна викликати функцію next(), щоб передати керування наступною проміжною програмою. Якщо ви передаєте аргумент, Express приймає його за помилку. Він пропустить всі функції middleware, що залишилися, які не обробляють помилки, і почне виконувати middleware, яка обробляє помилки.

Функції middleware не повинні мати значення return. Будь-яке значення, що повертається проміжним програмним забезпеченням, не буде використано Express.

Розглянемо типи існуючих middleware[2]. Звичайне проміжне програмне забезпечення (middleware). Більшість функцій Middleware, з які ми будемо використовувати в нашому додатку Express, будуть простими проміжним програмним забезпеченням і виглядатимуть як функції, розглянуті вище. Існує також middleware для обробки помилок. Різниця між middleware для обробки помилок та звичайним middleware полягає в тому, що функції middleware для обробки помилок задають чотири параметри замість трьох - (error, request, response, next).

```
function errorHandlerMiddleware(error, request, response, next) {
  console.log(error.message);
  next(error);
}
```

Вона буде викликана коли будь яка з інших проміжних функцій викинуть помилку, ця помилка і потрапить і обробник помилок як параметр

Використання middleware. Порядок налаштування middleware дуже важливий. Ми можемо застосувати їх на трьох рівнях у додатку:

1. Рівень маршруту.
2. Рівень маршрутизатора.
3. Рівень програми.

Для того щоб маршрут обробляв помилки, які він викликає, за допомогою middleware для обробки помилок, потрібно додати його після визначення маршруту.

Розглянемо кожен рівень окремо:

- на рівні маршруту. Це найконкретніший рівень: будь-яке проміжне програмне забезпечення, яке ми туди присвоїмо, буде працювати тільки для цього певного маршруту;
- на рівні маршрутизатора. Express дозволяє створювати об'єкти Router. Вони дозволяють обмежити використання middleware певним набором маршрутів. Якщо потрібно, щоб один і той же middleware виконувався для декількох маршрутів, а не для всіх, ці об'єкти саме те що треба;
- на рівні програми. Це найменш конкретний рівень. Будь-яке проміжне програмне забезпечення, налаштоване на цьому рівні, буде запущене для всіх існуючих маршрутів.

Налаштовувати middleware на рівні програми варто тільки у разі нагальної потреби, а саме, якщо його дійсно потрібно запускати для кожного маршруту у вашому додатку. Кожна функція middleware, незалежно від того, наскільки вона мала, потребує певного часу для виконання. Чим більше функцій middleware необхідно запустити для маршруту, тим повільніше виконуватимуться запити до цього маршруту.

2.4 Бази даних

Розглянемо частину програмної реалізації бази даних.

Розглянемо два можливих варіанти для збереження даних. Першим засобом візьмемо NoSQL базу даних MongoDB, альтернативою буде PostgreSQL. Доцільність використання однієї з них буде перевірена та обрана в ході розробки та тестування.

PostgreSQL[5]. Класична реляційна СУБД з покращеннями у вигляді швидшої роботи, можливості більшого масштабування та взаємодією з широким спектром різних платформ та технологій. Варто приділити більше уваги більш революційній та новій документоорієнтованій СУБД, тому перейдемо до MonogoDB.

2.4.1 MongoDB

Даний підрозділ використовує опис, наведений в [4].

MongoDB є найбільш популярною на даний момент документоорієнтованою системою управління базами даних.

MongoDB для зберігання даних використовує документи. На відміну від рядків, які зберігаються у реляційних базах даних, документи можуть зберігати складніші структури інформації. Документ можна представити як сховище ключів і значень даних, з якими цей ключ асоційований.

У реляційних СУБД існує поняття первинного ключа, яке відповідає певному стовпцю унікальних значень відносно кожного рядка. У MongoDB

для кожного документа є унікальний ідентифікатор id – створюється автоматично базою даних, якщо не вказати явно.

Кожному ключу надається визначене значення. Але слід враховувати що якщо в реляційних базах є чітко описана структура, де є поля, і якщо якесь поле не має значення, йому можна задати базове значення NULL, то в MongoDB такий ключ просто упускається в документі і не використовується далі.

У реляційних базах даних по типу SQL сутністю вищою ніж рядки є таблиці, а документоорієнтованій MongoDB є колекції, які володіють набором документів. Проте у реляційних таблицях об'єкти мають бути жорстко структорованимим для збереження, а в колекціях можуть міститися об’єкти, з різним структурою та набором властивостей.

Формат даних в MongoDB. Одним із популярних стандартів обміну даними та їх зберігання є JSON (JavaScriptObjectNotation). JSON ефективно описує складні структури даних. Спосіб зберігання даних у MongoDB в цьому плані схожий на JSON, хоча формально JSON не використовується. Для зберігання в MongoDB використовується формат, який називається BSON (Binary JSON).[17]

BSON дозволяє працювати з даними швидше: швидше виконується пошук і обробка. Хоча слід відзначити, що BSON на відміну від зберігання даних у форматі JSON має невеликі недоліки: у всіх даних у форматі JSON займають менше місця, ніж у форматі BSON, з іншого боку, цей недолік виливається у більшій швидкості взаємодії.

Простота у використанні. Відсутність жорсткої схеми бази даних і відповідно відсутність необхідності перестворювати цю схему при зміні концепції зберігання даних значно спрощує роботу з базами даних MongoDB і подальшим їх масштабуванням.

Але, навіть враховуючи всі недоліки традиційної бази даних і переваги MongoDB, важливо враховувати, що різні завдання відповідають різним підходам рішення. MongoDB може покращити продуктивність, наприклад,

якщо потрібно зберігати складні за структурою дані. Існують ситуації коли краще буде використовувати традиційні реляційні бази даних.

2.5 Висновки до розділу 2

У цьому розділі було розглянуто інструменти які будуть використані у розробці клієнтської та серверної частин додатку. Було розглянуто два варіанти різних за типом баз даних.

В рамках розгляду серверної частини було описано бібліотеку загального доступу express та загальні питання теорії про запити у мережі Інтернет. Також було розглянуто поняття проміжного програмного забезпечення.

Щодо клієнтської частини, була розглянута сучасна бібліотека для створення користувацьких інтерфейсів односторінкового типу React. Також було описано механізми бібліотеки, компоненти та методи які надають додатку життя.

Щодо баз даних, під розгляд потрапили документоорієнтована MongooseDB та реляційна PostgreSQL. Були описані недоліки та переваги обох, а також умови доцільності використання.

РОЗДІЛ 3
РОЗРОБКА ДОДАТКУ

3.1 Підготовка серверної частини

Для початку створення додатку, було запущено скрипт ініціалізації node.js проекту npm run init, за допомогою вбудованого в ядро npm – node package manager – менеджера пакунків node.js ядра.

Автоматично було створено package.json файл, що є основою проекту: тут зберігаються усі дані, пакунки, власник та скрипти. Надалі були встановлені за допомогою npm наступні пакунки:

1. "@mui/material" – буде використаний для зовнішнього оформлення таблиць, полів уведень, форм.
2. "bcrypt" – використовується для хешування, шифрування, порівняння строк(паролів).
3. "config" – дозволяє додавати у середовище проекту змінні, які використовуються для налаштування додатку, чи локальної розробки, також може зберігати та передавати необхідні змінні.
4. "create-react-app"- для створення інтерфейс додатку, буде розглядатися далі, як окремий, але дочірній проект серверної частини;
5. "express" – пакунок для роботи створеного, за допомогою нього ж, АПІ – сервісу.
6. "express-validator" – для валідації та перевірки даних, які потраплятимуть у АПІ – сервіс.
7. "jsonwebtoken" – для створення унікального токену користувача, який забезпечить коректну роботу додатку, у поточній сесії використання, після здійснення входу.

8. "mongoose" – для взаємодії з базою даних, та створення схем, для передачі до неї даних.

Почнемо створювати та налаштовувати АПІ – сервіс. На етапі розробки будемо слухати його на 5000 порті локальної машини. Створимо екземпляр бази даних, використовуючи веб-сервіс mongoose, звідки, по завершенню налаштувань, отримаємо URI для під'єднання. Зберігати URI бази даних будемо у config(folder)/default.json файлі який витягуватиме цю строку за допомогою config пакунку.

З огляду на необхідність сесійної роботи додатку, було вирішено добавити етап авторизації користувача по шляху /api/auth. Визначимо дві кінцеві точки /login та /register, для створення запитів для реєстрація та входу користувача. Обидві будуть використовувати метод запиту POST.

Розглянемо /register. Отримуючи дані з інтерфейсу, сервіс перевірятиме їх на коректність, та добавлятиме користувача у базу даних, якщо такий, не був створений раніше. Перевірка виконується за допомогою check та validationResult функцій у вигляді middleware методів. Пароль буде потрапляти у базу даних у вигляді зашифрованої строки – шифрування відбувається за допомогою бібліотеки bcrypy. Модель для передачі даних у базу буде створена за допомогою mongoose: поштова адреса буде вимагатися унікальним – за допомогою використання вбудованого конструктора класу Schema та окремого методу model для прив'язки схеми до її назви User, яка, до речі, буде відповідно відображатися у документах бази даних. У випадку успішної роботи повернеться 201 код, якщо ні – помилка 500 та повідомлення. Якщо ж користувач з такою поштовою скринькою вже існує – повернеться 400 помилка з відповідним текстом.

Розглянемо /login. Працює схожим чином, відмінністю є пошук користувача по поштовій скриньці у базі даних. А також повернення унікального зашифрованого токену, з обмеженим часом існування для активної сесії користувача. Токен створюється за допомогою jsonwebtoken пакунку, за допомогою використання секретного слова, яке знаходиться у

файлі конфігурації проекту також у токені буде зашифровано інформацію про користувача: його id. Це слово витягується динамічно під час здійснення запиту з config/default.json файлу. Час життя даного токену, для комфортної роботи, попередньо було обрано на 3 години – цей параметр додатку легко змінити. Модель користувача звісно використовується така ж як і для реєстрації. При здійсненні входу перевіряється пароль користувача. Якщо запит успішний – сервером повертається тіло з вмістом токену користувача та унікальним id(створюється і додається автоматично базою даних при реєстрації).

Як і для перевірки існування користувача при реєстрації чи створення нового документу зареєстрованого користувача , так і для співставлення уведених даних при вході використовуються вбудовані у mongoose функції findOne та save, які є методами попередньо створених моделей. При створенні моделі, беремо схеми які створювалися раніше та передаємо дані у конструктор моделі, із нашого запиту - тут у нас це User, після чого буде створений екземпляр класу User(наша модель).

Для перевірки чи токен користувача ще не просрочений(3 години) – було створено кінцеву точку /token. Перевірка відбувається за допомогою middleware auth. Із заголовку запиту дістається рядок авторизації та перевіряється на актуальність. Якщо все гаразд, тоді повертаємо 201 код та відповідне повідомлення, якщо ж рядка немає або час життя токену сплив тоді 401 код та пишемо що користувача не авторизовано чи час життя токену сплив.

Перейдемо безпосередньо до розгляду даних та кінцевих точок, які мають стосунок до роботи логіки двофакторного методу морфологічного аналізу. Було прийнято рішення оперувати трьома сутностями. InitialInput. Це буде масив об’єктів з полями:

1. parameter – тип String – назва параметру;
2. alternatives – тип масив String – масив назв альтернатив параметру;

3. alternativesValues – тип масив String – масив значень уведених альтернатив.

Усі поля вважаються необхідними для уведення. Дана схема відповідає введенню початкових даних користувачем: параметри, альтернативи та значення альтернатив. Також особливістю є alternativesValues, оскільки тут за значення будуть узяті числові значення перетворені у строку, це здійснено для подальшої роботи з даними та уніфікації методів. RelationsInput. Це буде об’єкт з полями:

1. finalRes – масив об’єктів з полями:

– baseAltName – тип String – назва альтернативи взята за основу для відслідковування взаємозв’язку з іншою альтернативою;

– relatedAlt– тип String – альтернатива взята для відслідковування взаємозв’язку з першим полем;

– value– тип Number –значення взаємозв’язку першого і другого параметрів;

2. dataId– тип Types.ObjectId(вбудований тип mongoose для відслідковування id значень документів) – ідентифікаційний номер об’єкту початкових даних до якого відноситься ця сутність з альтернативами та значеннями взаємозв’язків. Усі поля вважаються необхідними для уведення. Дана схема відповідає уведеним значенням взаємозв’язків альтернатив параметрів.

CombinationsProbability. Це буде об’єкт з полями:

1. combinationsProbability – тип масив об’єктів з полями:

– combination – тип масив String – масив з переліком комбінацій альтернатив;

– probability – тип Number – значення ймовірності даної комбінації альтернатив;

2. dataId– тип Types.ObjectId(вбудований тип mongoose для відслідковування id значень документів) – ідентифікаційний номер

об’єкту початкових даних до якого відноситься ця сутність з комбінаціями та ймовірностями. Уведений для зв’язування початкових даних, та усіх наступних пов’язаних з ними. Це також використовуватиметься для розгалуження роботи користувача, для можливості роботи з декількома система водночас.

Усі поля вважаються необхідними для уведення. Дана схема відповідає комбінаціям альтернатив та відповідним ймовірностям.

Перейдемо до кінцевих точок де використовуються ці схеми. Кінцева точка /initial.

Використовуються із запитами POST та GET. POST запит використовується для створення початкових даних та, якщо успіх , повертає відповідне повідомлення та id створених даних. GET запит використовується для отримання усіх початкових даних, фільтруючи їх по користувачу.

Кінцева точка /initial/:id. Використовуються із запитом GET. GET запит використовується для отримання початкових даних, виходячи з id, який передається у вигляді параметру запиту.

Кінцева точка /initial/ relations. Використовуються із запитом POST. POST запит використовується для створення даних про взаємозв’язок альтернатив, прив’язуючи dataId поле до цих даних, яке відповідає певним початковим даним. Якщо дані з такою прив’язкою уже є, вони перестворюватимуться новими. Якщо успіх, повертає відповідне повідомлення та створені дані.

Кінцева точка /initial/ relations/:id. Використовуються із запитом GET. GET запит використовується для отримання даних про взаємозв’язок альтернатив, фільтруючи дані по dataId полю прив’язаному до цих даних, для отримання екземпляру який відповідає певним початковим даним.. Якщо успіх, повертає відповідні дані.

Кінцева точка /initial/ combinations. Використовуються із запитом POST. POST запит використовується для створення даних про комбінації альтернатив, та їхні відповідні ймовірності, прив’язуючи dataId поле до цих

даних, яке відповідає певним початковим даним та даним про взаємозв’язок альтернатив. Якщо дані з такою прив’язкою уже є, вони перестворюватимуться новими. Якщо успіх, повертає відповідне повідомлення.

Кінцева точка /initial/combinations/:id. Використовуються із запитом GET. GET запит використовується для отримання даних про комбінації альтернатив, фільтруючи дані по dataId полю прив’язаному до цих даних, для отримання екземпляру який відповідає певним початковим даним та даним про взаємозв’язок альтернатив.. Якщо успіх, повертає відповідні дані.

Що стосується усіх кінцевих точок, то при успіху відправлятиметься 201 код та, можливо, дані. У випадку помилок 500 код. Також у кожному запиті використовується раніше згаданий middleware “auth”, який використовується для перевірки токену та часу його життя, тобто при усіх запитах із сервісу перевірятиметься токен. Також логікою передбачено розшифрування токену на стороні АПІ-сервісу та витягування звідти даних про користувача: його id, який, власне використовуватиметься у подальших обробках у кінцевих точка для отримання даних по користувачу

3.2 Підготовка клієнтської частини

Для початку роботи з інтерфейсом налаштуємо React проект. Почнемо із запуску вбудованого скрипту create-react-app в дочірній головній папці проекту. Це автоматично створить керуючий файл, та файл запуску алгоритму на HTML сторінці.

Надалі встановлюємо за допомогою npm наступні пакунки:

1. "@emotion/react" – для роботи стилів mui пакунку у React середовищі.
2. "@emotion/styled" – стилізація для mui пакунку.

3. "@mui/material" – основний пакунок для створення елементів інтерфейсу, використовуючи його оригінальні сутності.
4. "materialize-css" – допоміжний пакунок для створення елементів на інтерфейсі, містить вбудовані стилі, які використовуються за класовою схемою вбудованих стилів елементів.
5. "react" – основоположна бібліотека для запуску, ініціалізації та функціонування React додатку.
6. "react-dom" – для коректної роботи та розташування елементів на розмітці сторінки.
7. "react-router-dom" – для створення шляхів нас сторінці.

Наступним етапом буде створення компонентів. Почнемо зі шляхів, добавимо для уведення даних, перегляду результатів та авторизації. Кожному з ним буде відповідати своя сторінка, які розглянемо пізніше. Сторінка авторизації має наступний вигляд (див. рисунок 3.1).

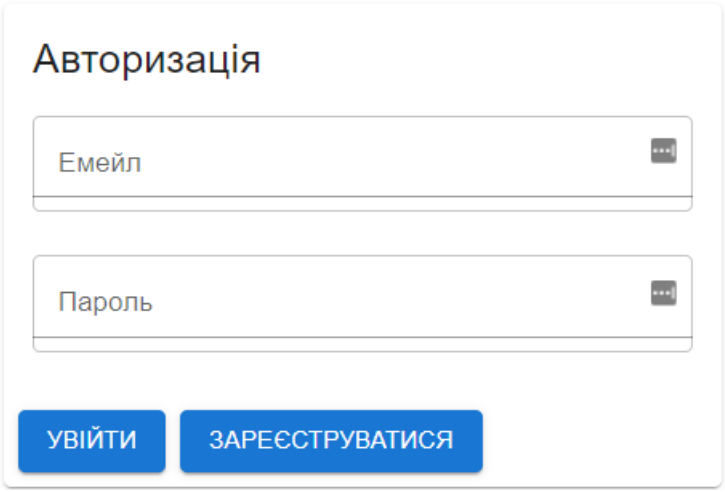


Рисунок 3.1 – Сторінка авторизації

При уведенні даних та натиску на кнопки буде відправлятися запит у АПІ сервіс.(/login чи /register відповідно). Дані мають бути коректні для успішного запиту, якщо ні відповідні спливаючі повідомленні будуть відображенні

До речі, React додаток буде запущено на порті 3000 локальної машини, тому у package.json вказуємо проксі 5000, щоб запити відправлялися куди потрібно.

Після успішної реєстрації, а потім входу у додаток переходимо на головну сторінку уведення даних (див. рисунок 3.2).

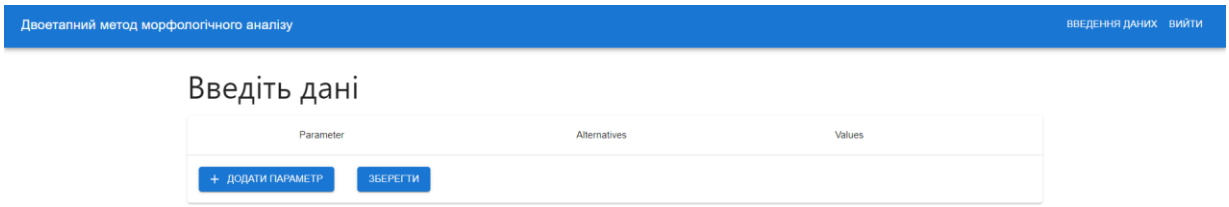


Рисунок 3.2 – сторінка уведення даних

Розглянемо навігаційну панель. При натиску на текст ліворуч, відбувається перенаправлення на домашню сторінку – уведення даних. Праворуч відповідно буде навігація на цю ж сторінку, у випадку уведення даних також з’явиться навігація на наступні етапи процесу, крайня справа навігація дає можливість вийти із додатку(після цього деактивується токен, і потрібно буде входити заново).

Розглянемо таблицю вводу. Після натискання відповідної кнопки «Додати параметр» будуть створені поля для уведення параметрів у таблиці, для додавання(кнопка) та уведення альтернатив та значень (див. рисунок 3.3).

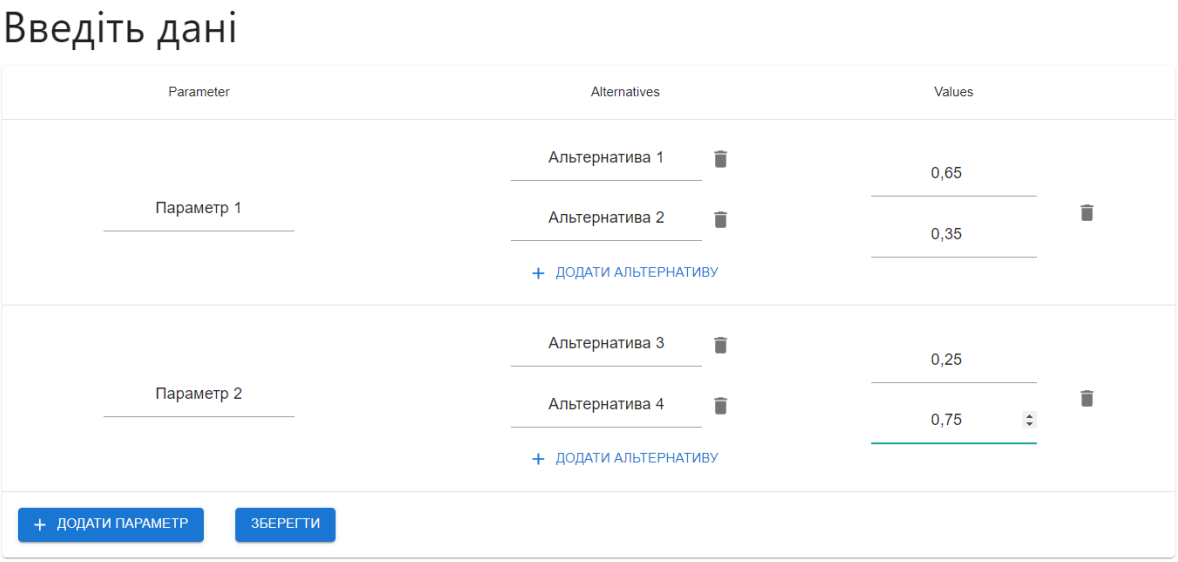


Рисунок 3.3 – уведення даних

Додатково є можливість прибрати зайві введення. Після уведення параметрів та альтернатив при натиску на кнопку «ЗБЕРЕГТИ», додаються створюється відповідний запит на створення цих даних(/api/input/initial) та задається значення dataId, яке рівне значенню id створеного документу. Також відобразиться можливість уведення матриці взаємозв’язків (див. рисунок 3.4).

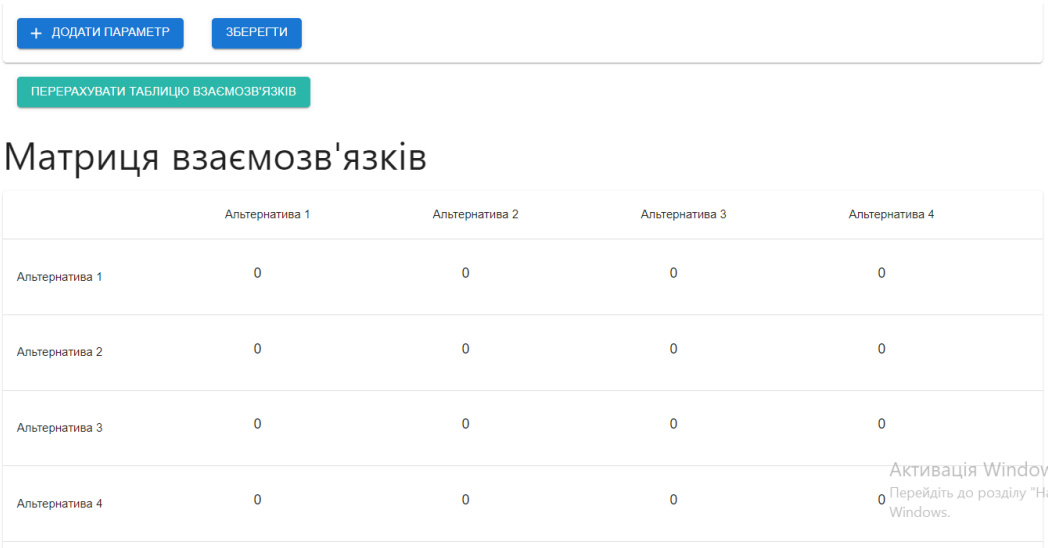


Рисунок 3.4 – матриця взаємозв’язків

Після натиску кнопки «ПЕРЕРАХУВАТИ МАТРИЦЮ ВЗАЄМОЗВ’ЯЗКІВ» з’являється вищепоказана матриця, куди можна увести

дані, після уведення даних з’являється кнопка «ОЦІНКИ АЛЬТЕРНАТИВ». Також при натиску кнопки витягуються дані про останній вхідні дані звідки формується таблиця - /api/input/initial/dataId (див. рисунок 3.5).

Рисунок 3.5 – уведення даних в матрицю взаємозв’язків

У випадку зміни будь якого з вхідних значень – матриця буде прибрана з інферфейсу й потрібно буде знову зберегти дані а потім розрахувати нову матрицю, після чого це вже буде зовсім інший екземпляр нової системи.

Після натиску кнопки «ОЦІНКИ АЛЬТЕРНАТИВ» користувача буде здійснено запит на створення даних про взаємозв’ящки альтернатив(api/input/initial/relations з dataId у тілі запиту) та перенаправлено на нову сторінку (див. рисунок 3.6) з ймовірностями альтернати разом з ймовірностями та оцінками. Причому перенаправлення відбудеться по посиланню з параметром dataId, тобто ця сесія буде іти паралельно разом з іншими, якщо інші початкові дані було створено – це дозволяє працювати з різними системами паралельно.

Приклад посилання:
http://localhost:3000/create/647cb0d00a38e7079b8d69f0

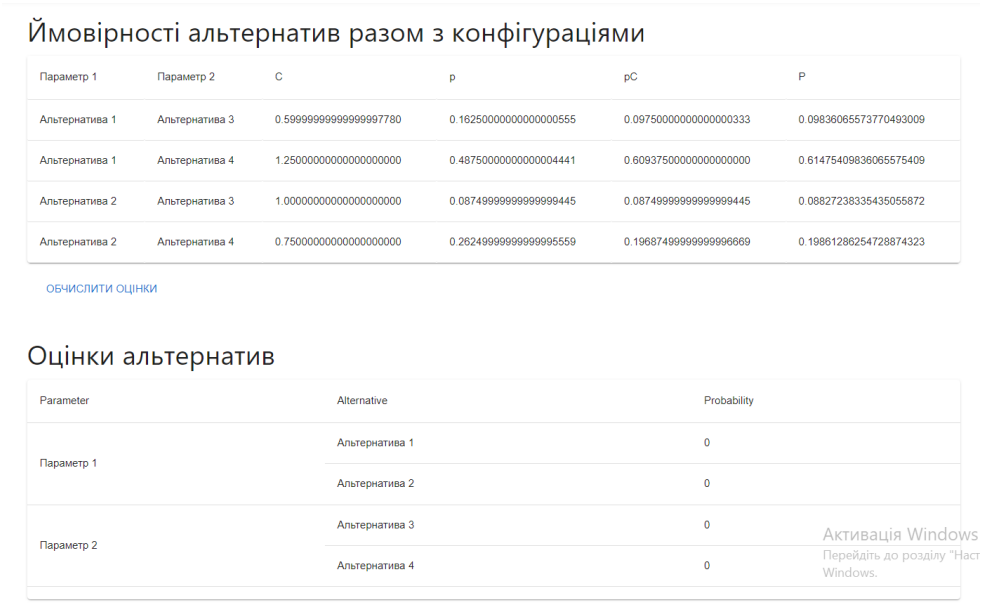


Рисунок 3.6 – сторінка результатів першого етапу

Після натискання кнопки «ОБЧИСЛИТИ ОЦІНКИ» будуть здійснені обчислені та оновлені значення таблиці «Оцінки альтернатив». Після натискання кнопки «ДРУГИЙ ЕТАП» буде створений запит на збереження даних про комбінації та відповідні ймовірності(/api/input/initial/combinations з dataId у тілі запиту, який витягуватиметься з актуального посилання сторінки) та перенаправлено користувача на сторінку другого етапу (див. рисунок 3.7) методу у посиланні якого також буде присутній параметр dataId .

Приклад посилання:

http://localhost:3000/second/647cb0d00a38e7079b8d69f0

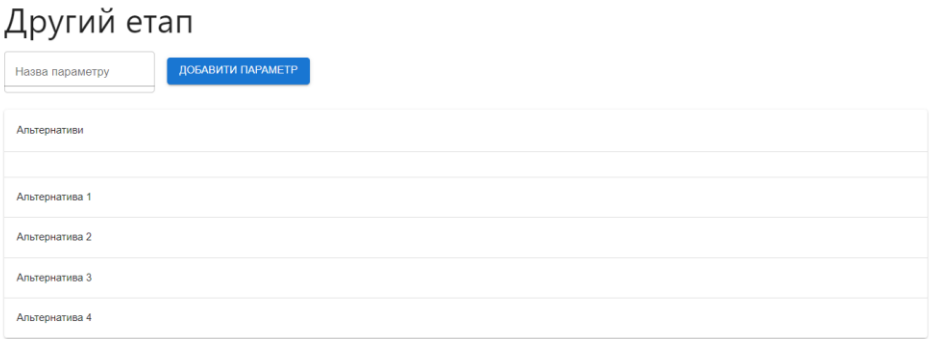


Рисунок 3.7 – сторінка другого етапу

Надається можливість додавати та видаляти як параметри так і альтернативи другого етапу і також відповідні значення (див. рисунок 3.8).

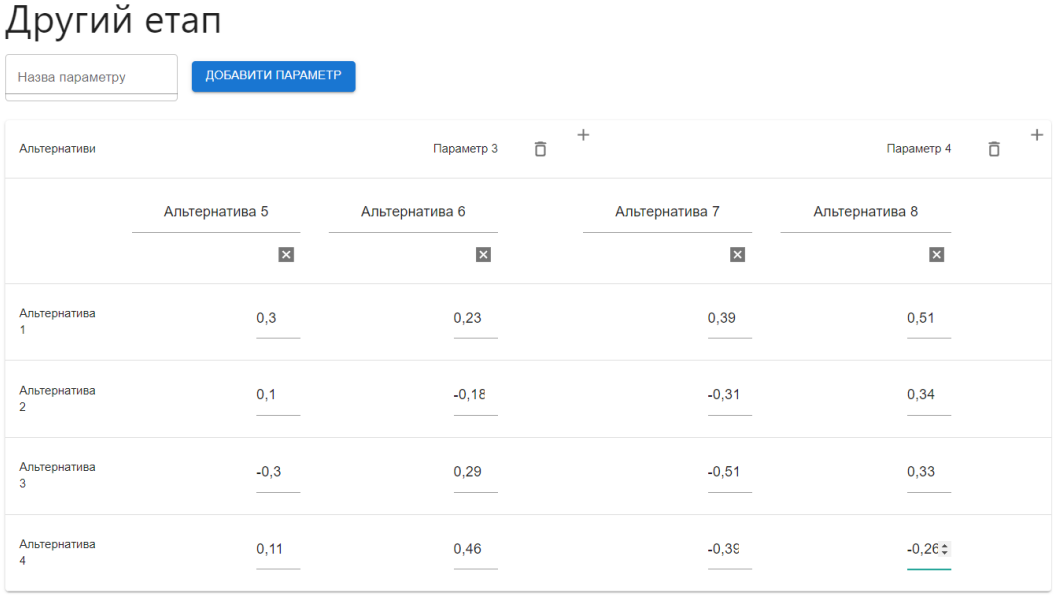


Рисунок 3.8 – уведення даних другого етапу

Після уведення даних нижче таблиці будуть виведені відповідні таблиці конфігурацій та очікуваних результатів по кожному параметру окремо (див. рисунок 3.9).

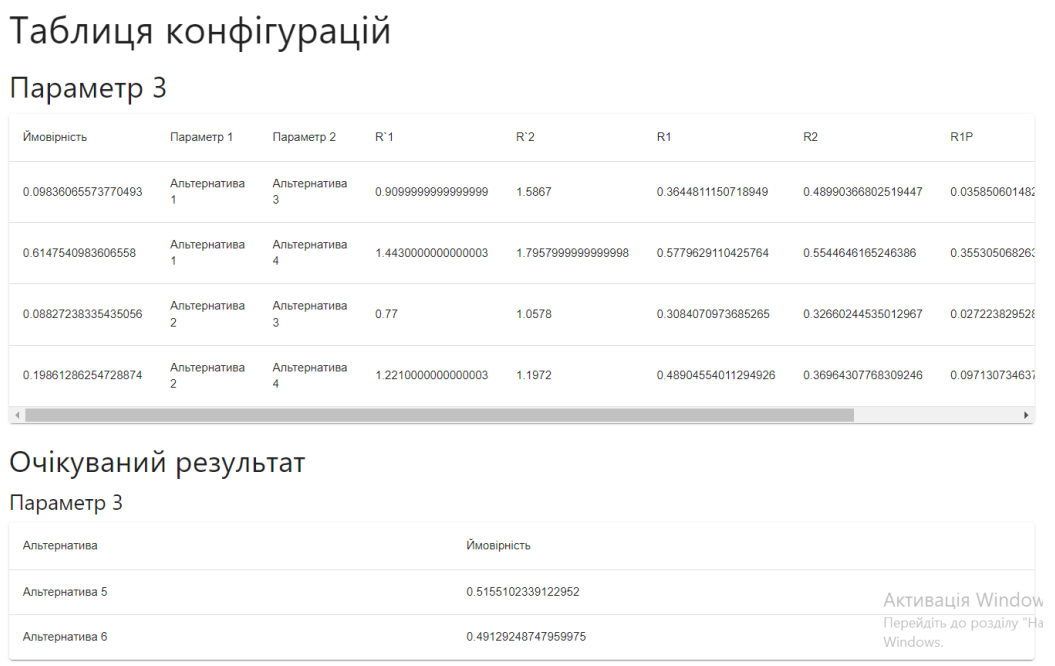


Рисунок 3.9 – результати другого етапу

3.3 Висновки до розділу 3

У цьому розділі було розглянуто процес розробки клієнтської та серверної частин додатку. Було розглянуто, яким чином клієнтська частина взаємодіє із серверною, які здійснює запити.

В рамках розгляду серверної частини було описано використані пакунки, як влаштований сам модуль, з яких частин складається. Детально було розглянуто роботу усіх кінцевих точок серверу.

Щодо клієнтської частини, був розглянутий інтерфейс, усі частини що взаємодіють з користувачем, і які запити та зміни відбуваються при взаємодії з різними елементами.

РОЗДІЛ 4
ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі буде проведено оцінювання основних характеристик для розгортання, публікації та підтримки програмного продукту, що спеціалізується розв’язанні задач сценарного аналізу за використання модифікованого методу морфологічного аналізу.

Впровадження такого сервісу забезпечить комфортну та зручну роботу із різноманітними задачами сценарного аналізу, сприятиме швидким обчисленням, ефективному аналізу та оцінці результатів

Також в даному дослідженні показано кілька варіантів розгортання та конфігурацій використаних для розробки інструментів, для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розгортання та підтримки середовища розв’язання задач сценарного аналізу. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для обробки вхідних даних, зберігання даних та проведення обчислень методом морфологічного аналізу.

Технічні вимоги до програмного продукту є наступні:

– функціонування у широкому спектрі браузерів;

– зручність та зрозумілість для користувача;

– швидкість обробки даних та доступ до інформації в реальному часі;

– можливість зручного масштабування та обслуговування;

– мінімальні витрати на впровадження програмного продукту;

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розгортання програмного продукту, який дозволяє проводити обчислення для розв’язання задачі сценарного аналізу. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір обчислювальної інфраструктури,

F_2 – вибір програмного каркасу для АПІ-сервісу,

F_3 – вибір бази даних.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) AWS,

б) Azur,

в) GCP.

Функція F_2.

а) express,

б) nest.

Функція F_3:

а) Mongoose,

б) PostgreSQL.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (див рисунок 4.1).

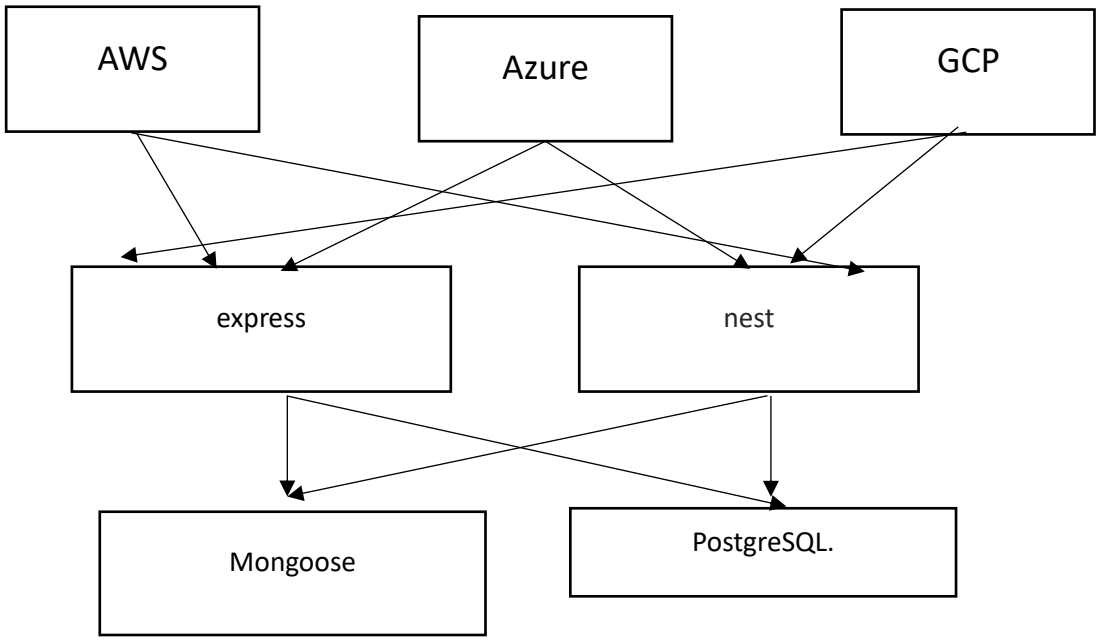


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантиів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Велика швидкість здійснення обчисень та відгуку	Складна у вивченні та реалізації
	Б	Легко інтегрується з іншими сервісами домену	Обчислення відбуваються довше
	В	Легкий у вивченні та користуванні	Функціонально обмежений
F_2	*А*	Легкий у використанні	Підтримує додатки з нескладною логікою
	Б	Складний у вивченні	Підтримує використання складної логіки
F_3	*А*	Реалізує складні зв’язки з різними сутностями	Складно масштабується
	Б	Структурована та інуїтивно зрозуміла	Важко масштабується при наявності складних зв’язків

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

– Функція F_1*:* перевагу даємо AWS. Для необхідної швидкості виконання та відгуку варіанти Б та В мають бути відкинуті,

– Функція F_2*:*Програма допускає обрання обох варіантів. Можливо використати варіанти А чи Б,

– Функція F_3*:* реалізація першого варіанту є сприйнятливою для програми. Це варіант А.

Таким чином, будемо розглядати такий варіанти використання інструментів ПП:

F_1а – F_2а – F_3а,

F_1а – F_2б – F_3а.

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- X1 – швидкодія здійснення обчислень,
- X2 – середнє значення об’єму пам’яті для обчислень та збереження даних,
- X3 – час відгуку АПІ-сервісу,
- X4 – час рендерингу сторінки.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 - Основні параметри програмного продукту

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія здійснення обчислень	X1	сек	2	1	0,5
Об'єм пам'яті	X2	Кб	6	4,5	3,5
Час відгуку сервісу	X3	сек	1	0,5	0,3
Час рендерингу сторінки	X4	сек	0,1	0,08	0,03

За даними таблиці 4.3 будуються графічні характеристики параметрів рисунок 4.2 – рисунок 4.5.

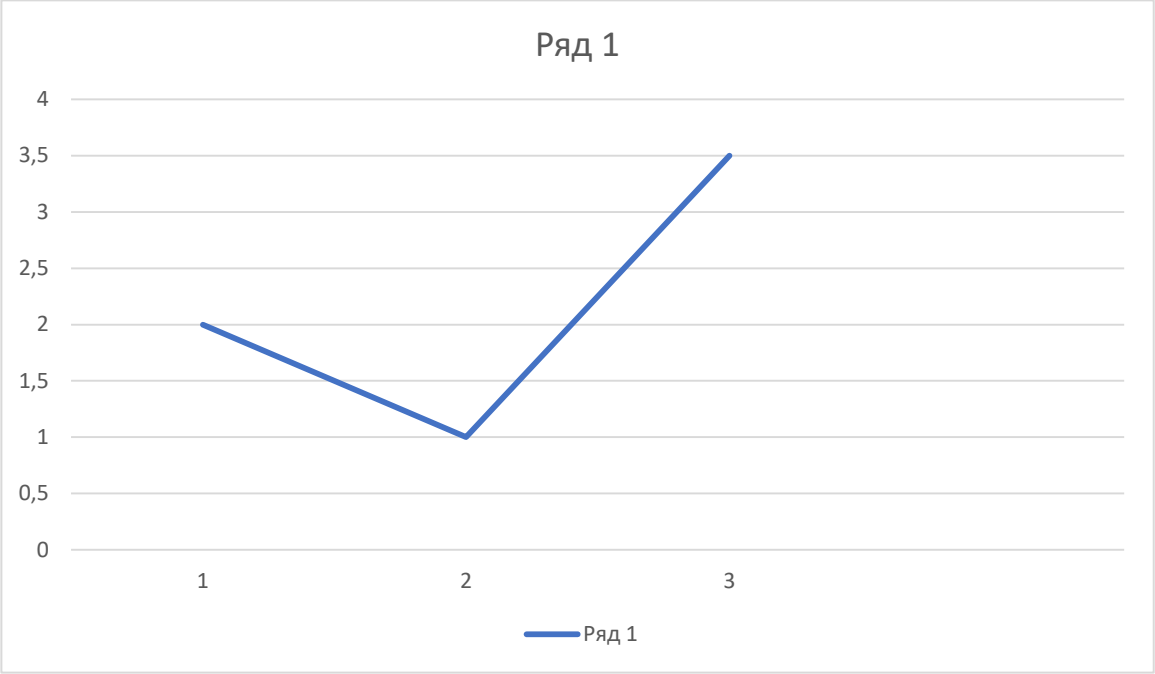


Рисунок 4.2 – X1, швидкодія обчислень

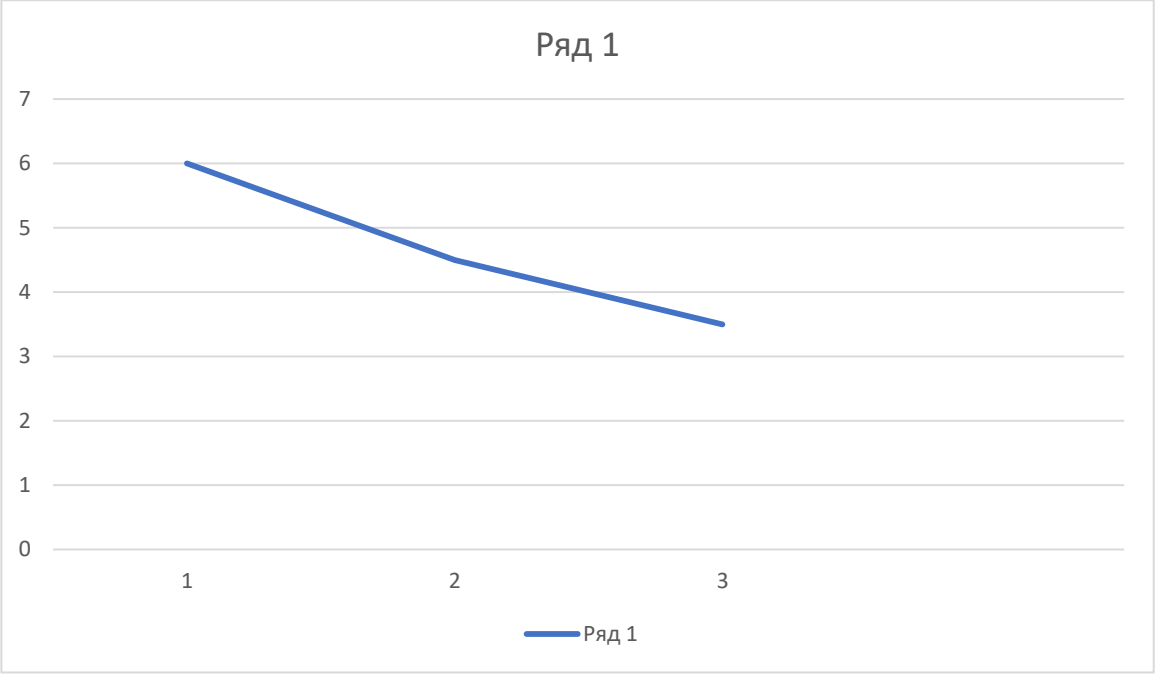


Рисунок 4.3 – X2, об’єм пам’яті

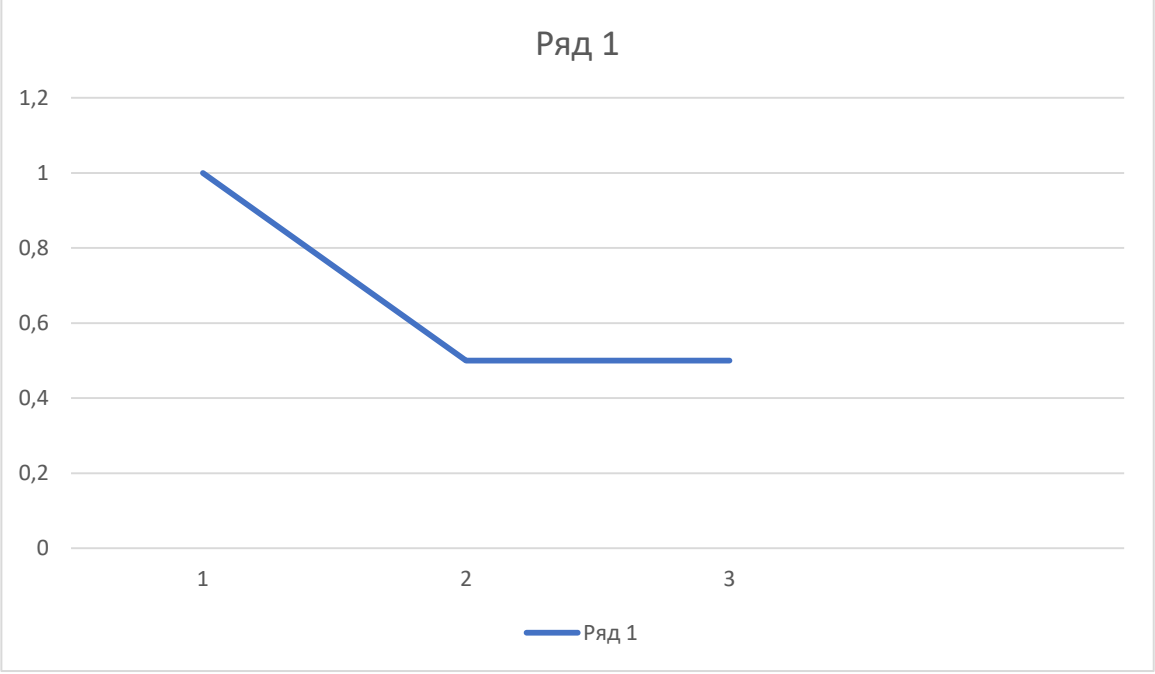


Рисунок 4.4 – X3, час відгуку сервісу

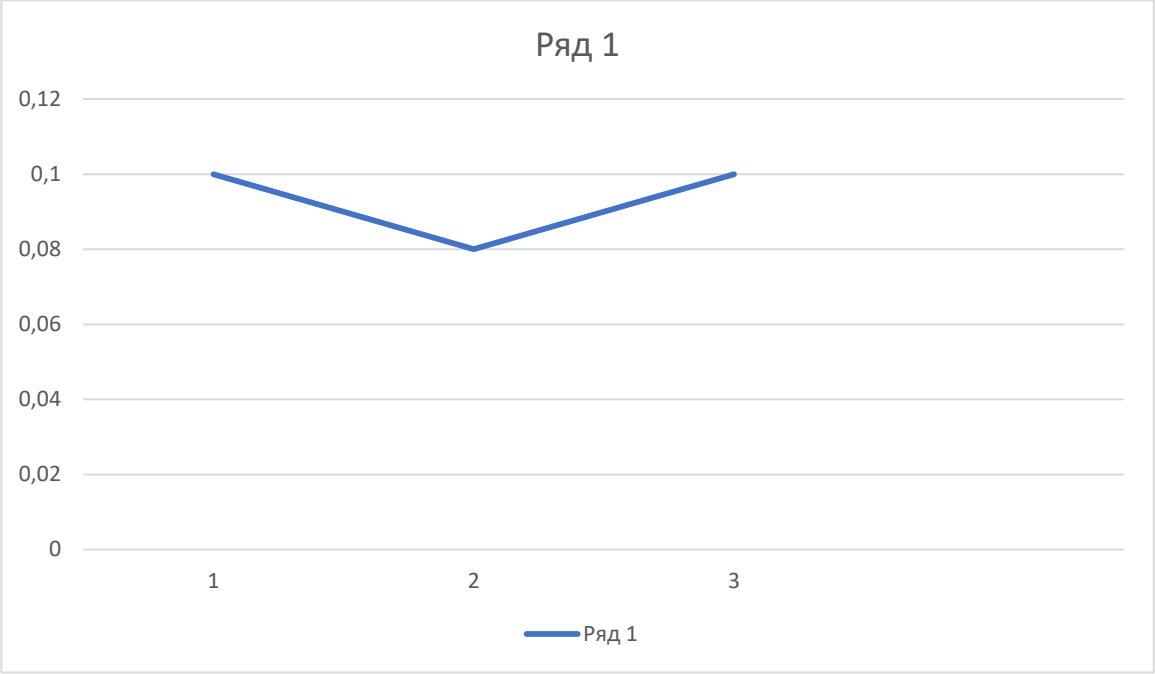


Рисунок 4.5 – X3, час рендерингу сторінки

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розгортання програмного продукту, який дозволяє виконувати сценарний аналіз на основі модифікованого методу морфологічного аналізу .

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів,

– перевірку придатності експертних оцінок для подальшого використання,

– визначення оцінки попарного пріоритету параметрів,

– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів Ri	Відхи-лення Δi	Δi2
			1	2	3	4	5	6	7			
X1	Швидкодія здйіснення обчислень	сек	1	1	2	1	1	2	1	9	-8,5	72,25
X2	Обєм пам'яті	Кб	3	4	3	4	3	3	4	24	6,5	42,25
X3	Час відгуку сервісу	сек	2	2	1	2	2	1	2	12	-5,5	30,25
X4	Час рендерингу сторінки	сек	4	3	4	3	4	4	3	25	7,5	56,25
	Разом		10	10	10	10	10	10	10	70	0	201

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \#$$

де N – число експертів,

n – кількість параметрів,

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5\#$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \#$$

Сума відхилень по всім параметрам повинна дорівнювати 0,

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 201\ \#$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 201}{7^2(4^3 - 3)} = 0{,}82 > W_k = 0{,}67.\ \#$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	<	<	<	<	<	0,5
X1 і X3	<	<	>	<	<	>	<	<	0,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	>	>	>	>	>	>	>	>	1,5
X2 і X4	<	>	<	>	<	<	>	<	0,5
X3 і X4	<	<	<	<	<	<	<	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases} .\#$$

З отриманих числових оцінок переваги складемо матрицю A=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \#$$

$$b_i = \sum_{i=1}^{N} a_{ij} \#$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \#$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \#$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				1 іт		2 іт		3 іт	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1	0,5	0,5	0,5	2,5	0,16	6,25	0,091	15,625	0,05
X2	1,5	1	1,5	0,5	4,5	0,28	20,25	0,29	91,125	0,29
X3	1,5	0,5	1	0,5	3,5	0,22	12,25	0,18	42,875	0,14
X4	1,5	1,5	1,5	1	5,5	0,34	30,25	0,44	166,375	0,5
Всього:					16	1	69	1	316	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об'єм пам'яті), *X3* (час відгуку сервісу) та *X4* (час малювання сторінки) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкодія обчислень) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\ , \#$$

де n – кількість параметрів,

$K_{вi}$– коефіцієнт вагомості i–го параметра,

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	0,7	2	0,05	0,1
F2	А	X4	0,07	4	0,5	2
	Б	X3	0,58	4	0,14	0,56
F3	А	X2	4,25	3	0,29	0,87

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \#$$

визначаємо рівень якості кожного з варіантів:

$K_{K1} = 0{,}1 + 2 + 0{,}87 = 2{,}97$,

$K_{K2} = 0{,}1 + 0{,}56 + 0{,}87 = 1{,}53$.

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості. та внесемо дані у таблицю 4.7.

Таблиця 4.7 — Розрахунок працездатності варіантів реалізації основних функцій

Основні функції	Варіант реалізації функції	Загальна норма, Тр	Коефіцієнт КП	Коефіцієнт, КСІ	Коефіцієнт, КБД	Коефіцієнт, КСТ	Розрахункова працездатність, Т (чол./днів)
F1	А	25	1,65	1,2	1,1	0,9	49,005
F2	А	18	1,65	1,4	1,3	0,9	48,65
	Б	23	1,65	1,4	1,3	0,9	62,2
F3	А	16	1,65	1,2	1,4	0,9	39,9

$Т_Р$ – трудомісткість розгортання ПП,

$К_П$ – поправочний коефіцієнт,

$К_{СІ}$ – коефіцієнт на складність використання зовнішньої інфраструктури,

$К_{БД}$ – коефіцієнт бази даних,

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм.

На основі даних табл. 4.7 визначимо загальну працездатність за кожним із варіантів і працездатність тестування.

Складемо трудомісткість відповідних завдань для кожного з чотирьох обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

1. F1-A-F2–A–F3–A,
2. F1-A-F2–Б–F3–A.

Тоді:

$T_1 = (49{,}005 + 48{,}65 + 39{,}9) \square 8 = 1100{,}44$ чол/год,

$T_2 = (49{,}005 + 62{,}2 + 39{,}9) \times 8 = 1208{,}84$ чол/год.

Очевидно, що вищу працездатність має другий варіант. В розгортанні беруть участь два програмісти з окладом 20000 грн. Визначимо середню зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} \text{ грн., \#}$$

де М – місячний оклад працівників,

T_m – кількість робочих днів тиждень,

t – кількість робочих годин в день.

$$С_ч = \frac{20000 + 20000}{2 \cdot 21 \cdot 8} = 119{,}04 \text{ грн.\#}$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д \text{, \#}$$

де $С_ч$– величина погодинної оплати праці програміста,

$Т_i$ – трудомісткість відповідного завдання,

$К_Д$ – норматив, який враховує додаткову заробітну плату(оберемо 1,1).

Зарплата розробників за варіантами становить:

– $С_{ЗП} = 119.04 \cdot 1100,44 \cdot 1.1 = 144096,015$ грн.,
– $С_{ЗП} = 119.04 \cdot 1208,84 \cdot 1.1 = 158290,345$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

– $С_{ВІД} = С_{ЗП} \cdot 0.22 = 144096,015 \cdot 0.22 = 31701,12$ грн.,
– $С_{ВІД} = С_{ЗП} \cdot 0.22 = 158290,345 \cdot 0.22 = 34823,88$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$). Так як одна ЕОМ обслуговує одного програміста з окладом 20000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 2 \cdot М \cdot К_З = 2 \cdot 20000 \cdot 0,2 = 8000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 8000 \cdot (1 + 0.2) = 9600 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 48960 \cdot 0,22 = 2112 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 15% та вартості ЕОМ – 30000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.2 \cdot 0.15 \cdot 30000 = 5400 \text{ грн.},$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача,

$К_А$– річна норма амортизації,

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.2 \cdot 30000 \cdot 0.05 = 1800 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.6 =$$

$$= 1118,4 \text{ години},$$

де $Д_К$ – календарна кількість днів у році,

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів,

$Д_Р$ – кількість днів планових ремонтів устаткування,

t –кількість робочих годин в день,

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1118,4 \cdot 0,4 \cdot 0,6 \cdot 4,87 = 1307,19 \text{ грн.}$$

де $N_С$ – середньо-споживча потужність приладу,

$К_З$– коефіцієнтом зайнятості приладу,

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 30000 \cdot 0,67 = 20100 \quad \text{грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \#$$

$$С_{ЕКС} = 9600 + 2112 + 5400 + 1800 + 1307,19 + 20100 = 40319,19 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 40319,19 / 1118,4 = 36,1 \quad \text{грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot Т, \#$$

– СМ = 36,1 · 1100,44 = 39725,884 грн.,

– СМ = 36,1 · 1208,84 = 43639,124 грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67, \#$$

$$С_Н = 144096,015 \cdot 0,67 = 96544,32 \text{ грн,}$$

$С_Н = 158290,345 \cdot 0,67 = 106054,531$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \#$$

– $С_{ПП} = 144096,015 + 31701,12 + 38845,5 + 96544,32 = 311186,96$ грн,

– $С_{ПП} = 158290,345 + 34823,88 + 42672,052 + 106054,531 = 341840,81$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_j} = К_{Кj} / С_{Фj}, \#$$

– $К_{ТЕР1} = 2,97 / 311186,96 = 9,54 \cdot 10^{-6}$,

– $К_{ТЕР2} = 1,53 / 341840,81 = 4,48 \cdot 10^{-6}$.

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 9,54 \cdot 10^{-6}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного

комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 9,54 \cdot 10^{-6}$.

Цей варіант реалізації програмного продукту має такі параметри:

– Вибір обчислювальної інфраструктури продукту – AWS,
– Побудова АПІ – сервісу на основі програмного каркасу express,
– Використання бази даних документорієнтованого типу Mongoose.

Ця комбінація розгортання додатку забезпечує швидку роботу додатку, оперативний відгук АПІ – сервісу та оптимізоване зберігання даних продукту.

4.8 Висновки до розділу 4

У цьому розділі було розглянуто функціонально-вартісний аналіз розгортання програмного продукту. Після визначення основних функцій програмного продукту, було виконано дослідження у вигляді функціонального та вартісного аналізу можливих комбінацій розгортання веб-сервісу та його складових.

В рамках аналізу було знайдено основні характеристичні параметри системи. Враховуючи результати та можливі варіанти реалізації розгортання даного сервісу, було обрано найкращим чином найбільш ефективна комбінація для розгортання веб-додатку.

ВИСНОВКИ

Модифікований метод морфологічного аналізу слугує якісним інструментом аналітики задач. Зокрема, коли потрібно опрацьовувати об’єкти, які можуть бути неточні, невизначені, неповні. Це провокує виникнення великої кількості альтернативних варіантів реалізації. Застосовуючи модифікований метод морфологічного аналізу, ми маємо змогу описати об’єкт, дослідити його та прийняти рішення враховуючи його невизначену конфігурацію.

Користувацький інтерфейс забезпечить комфортне та зручне використання даного методу для проведення обчислень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Meta open Source: Built-in React Hooks. URL: https://react.dev/reference/react
2. Strong Loop/IBM Using MiddleWare. URL: https://expressjs.com/en/guide/using-middleware.html#using-middleware
3. OpenJS Foundation Node.js v14.21.3 documentation. URL: https://nodejs.org/docs/latest-v14.x/api/documentation.html
4. MongoDB founded by Dwight Merriman, Eliot Horowitz and Kevin Ryan, MongoDB Manual by MongoDB Inc. URL: https://www.mongodb.com/docs/manual/
5. The PostgreSQL Global Development Group, Documentation. URL: https://www.postgresql.org/docs/
6. Material UI SAS, Material UI Overview. URL: https://mui.com/material-ui/getting-started/overview/
7. Remy Sharpe, Nodemon repository on Github. URL : https://github.com/remy/nodemon/
8. Mozilla Corporation’s MDN contributors, JavaScript articles. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript
9. An introduction to the NPM package manager by OpenJs Foundation. URL: https://nodejs.dev/en/learn/an-introduction-to-the-npm-package-manager/
10. Sophia Brandt, JavaScript Development: Run Concurrently article. URL: https://www.rockyourcode.com/javascript-development-run-concurrently/

11. Helpers and tips for npm run scripts by Michael Kuehnel. URL: https://michael-kuehnel.de/tooling/2018/03/22/helpers-and-tips-for-npm-run-scripts.html

12. npm, Inc, Introduction to packages and modules articles. URl: https://docs.npmjs.com/packages-and-modules/introduction-to-packages-and-modules

13. Single-page application (SPA) article by Ivy Wigmore. URL: https://www.techtarget.com/whatis/definition/single-page-application-SPA#:~:text=A%20single%2Dpage%20application%20(SPA,%2Dpage%20interface%20(SPI).

14. What is Express.js? by Codeacademy Team. URL: https://www.codecademy.com/article/what-is-express-js

15. І. О. Савченко. Методологічне і математичне забезпечення розв’язання задач передбачення на основі модифікованого методу морфологічного аналізу // Системні дослідження та інформаційні технології, 2011, №3, С. 18-28. К. : Київський політехнічний інститут ім. Ігоря Сікорського URL: https://ela.kpi.ua/bitstream/123456789/9257/1/02_Savch.pdf.

16. Manu A. How to Prepare for React Interviews – Front-End Technical Interview Guide 2022. URL: https://www.freecodecamp.org/news/prepare-for-react-technical-interviews/.

ДОДАТОК А. Лістинг програмного продукту

package.json

```
{
  "name": "app",
  "version": "1.0.0",
  "description": "MSS two-staged method application",
  "main": "app.js",
  "scripts": {
    "start": "node app.js",
    "server": "nodemon app.js",
    "client": "npm run start --prefix client",
    "dev": "concurrently \"npm run server\" \"npm run client\""
  },
  "keywords": [
    "mss",
    "diploma"
  ],
  "author": "Ostap-Tadey Demchyshyn <forgotmey@gmail.com>",
  "license": "ISC",
  "dependencies": {
    "@mui/material": "^5.13.2",
    "bcrypt": "^5.1.0",
    "config": "^3.3.9",
    "create-react-app": "^5.0.1",
    "express": "^4.18.2",
    "express-validator": "^7.0.1",
    "jsonwebtoken": "^9.0.0",
    "mongoose": "^7.2.0"
  },
  "devDependencies": {
    "concurrently": "^8.0.1",
    "nodemon": "^2.0.22"
  }
}
```

App.js

```
const express = require('express')
const config = require('config')
const { default: mongoose } = require('mongoose')
const authRouter = require('./routes/auth.routes')
const inputRouter = require('./routes/input.routes')
const tokenRouter = require('./routes/token.routes')

const app = express()

app.use(express.json({ extended: true })) //for correct parsing requests body

app.use('/api/auth', authRouter)
app.use('/api/input', inputRouter)
app.use('/api', tokenRouter)

const PORT = config.get('port') | 5000

async function start() {
  try {
    await mongoose.connect(config.get('mongoUri'), {})
    app.listen(PORT, () => console.log(`app is working on ${PORT}`))
  } catch (e) {
    console.error(e.message)
    process.exit(1)
```

```
  }
}
start()

auth.routes.js
const { Router } = require('express')
const User = require('../models/user.model')
const router = Router()
const bcrypt = require('bcrypt')
const { check, validationResult } = require('express-validator')
const jwt = require('jsonwebtoken')
const config = require('config')

router.post(
  '/register',
  [
    check('email', 'Неправильний емейл').isEmail(),
    check('password', 'Мінімальна довжина пароля 6 символів').isLength({
      min: 6,
    }),
  ],

  async (req, res) => {
    try {
      const errors = validationResult(req)

      if (!errors.isEmpty()) {
        return res.status(400).json({
          errors: errors.array(),
          message: 'Неправильні дані',
        })
      }

      const { email, password } = req.body

      const candidate = await User.findOne({ email })
      if (candidate) {
        res.status(400).json({ message: 'Такий користувач існує' })
      } else {
        const hashedPassword = await bcrypt.hash(password, 2)
        const user = new User({ email, password: hashedPassword })
        await user.save()
        res.status(201).json({ message: 'Користувач створений' })
      }
    } catch (e) {
      res.status(500).json({ message: e.message })
    }
  }
)

router.post(
  '/login',
  [check('email', 'Введіть норм емейл').normalizeEmail().isEmail()],
  async (req, res) => {
    try {
      const errors = validationResult(req)

      if (!errors.isEmpty()) {
        return res.status(400).json({
          errors: errors.array(),
          message: 'Неправильні дані',
        })
      }

      const { email, password } = req.body

      const user = await User.findOne({ email })

      if (!user) {
        return res.status(400).json({ message: 'Користувача не знайдено' })
```

```
      }

      const isMatch = await bcrypt.compareSync(password, user.password)
      if (!isMatch) {
        return res.status(400).json({
          message: 'Поганий пароль',
          entered: password,
          haveToBe: user.password,
        })
      }

      const token = jwt.sign(
        {
          userId: user.id,
        },
        config.get('jwtSecret'),
        { expiresIn: '3h' }
      )
      res.json({ token, userId: user.id })
    } catch (e) {
      res.status(500).json({ message: 'Something went wrong' })
    }
  }
)

module.exports = router

input.routes.js
const { Router } = require('express')
const InitialInput = require('../models/initialInput.model')
const RelationsInput = require('../models/relationsInput.model')
const CombinationsProbability = require('../models/combinations.model')
const auth = require('../middleware/auth.middleware')

const router = Router()

router.post('/initial', auth, async (req, res) => {
  try {
    const input = new InitialInput({
      inputArray: req.body.rows,
      owner: req.user.userId,
    })
    await input.save()
    res
      .status(201)
      .json({ message: 'Введені дані збережені', dataId: input._id })
  } catch (e) {
    res.status(500).json({ message: e.message })
  }
})

router.get('/initial/:id', auth, async (req, res) => {
  try {
    const data = await InitialInput.findById(req.params.id)
    res.status(201).json({ data })
  } catch (e) {
    res.status(500).json({ message: e.message })
  }
})

router.get('/initial', auth, async (req, res) => {
  try {
    const data = await InitialInput.find({ owner: req.user.userId })
    res.status(201).json({ data })
  } catch (e) {
    res.status(500).json({ message: e.message })
  }
})

router.post(
```

```
  '/initial/relations',

  async (req, res) => {
    try {
      const exists = await RelationsInput.exists({ dataId: req.body.dataId })
      if (exists != null) {
        await RelationsInput.findOneAndRemove({ dataId: req.body.dataId })
      }
      const input = new RelationsInput({
        finalRes: req.body.finalRes,
        dataId: req.body.dataId,
      })
      res.status(201).json({ input })
      await input.save()
    } catch (e) {
      res.status(500).json({ message: e.message })
    }
  }
)

router.get('/initial/relations/:id', auth, async (req, res) => {
  try {
    const data = await RelationsInput.findOne({ dataId: req.params.id })
    res.status(201).json({ data })
  } catch (e) {
    res.status(500).json({ message: e.message })
  }
})

router.post(
  '/initial/combinations',

  async (req, res) => {
    try {
      const exists = await CombinationsProbability.exists({
        dataId: req.body.dataId,
      })
      if (exists != null) {
        await CombinationsProbability.findOneAndRemove({
          dataId: req.body.dataId,
        })
      }
      const input = new CombinationsProbability({
        combinationsProbability: req.body.combinationsProbability,
        dataId: req.body.dataId,
      })
      await input.save()
      res.status(201).json({ message: 'Введені дані збережені' })
    } catch (e) {
      res.status(500).json({ message: e.message })
    }
  }
)

router.get(
  '/initial/combinations/:id',

  async (req, res) => {
    try {
      const data = await CombinationsProbability.findOne({
        dataId: req.params.id,
      })
      res.status(201).json({ data })
    } catch (e) {
      res.status(500).json({ message: e.message })
    }
  }
)

module.exports = router
```

token.routes.js

```
const { Router } = require('express')
const auth = require('../middleware/auth.middleware')

const router = Router()

router.get('/token', auth, async (req, res) => {
  try {
    res.status(201).json({ message: 'Час життя токену не сплив' })
  } catch (e) {
    res.status(500).json({ message: e.message })
  }
})

module.exports = router
```

user.model.js

```
const { Schema, model, Types } = require('mongoose')

const schema = new Schema({
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
})

module.exports = model('User', schema)
```

initial.input.js

```
const { Schema, model, Types } = require('mongoose')

const initialSchema = new Schema({
  inputArray: [
    {
      parameter: {
        type: String,
        required: true,
      },
      alternatives: {
        type: [String],
        required: true,
      },
      alternativesValues: {
        type: [String],
        required: true,
      },
    },
  ],
  owner: { type: Types.ObjectId, ref: 'User' },
})

module.exports = model('InitialInput', initialSchema)
```

relationsInput.model.js

```
const { Schema, model, Types } = require('mongoose')

const initialSchema = new Schema({
  inputArray: [
    {
      parameter: {
        type: String,
        required: true,
      },
      alternatives: {
        type: [String],
        required: true,
      },
      alternativesValues: {
        type: [String],
        required: true,
```

```
      },
    },
  ],
  owner: { type: Types.ObjectId, ref: 'User' },
})

module.exports = model('InitialInput', initialSchema)

combinations.model.js
const { Schema, model, Types } = require('mongoose')

const combinationsSchema = new Schema({
  combinationsProbability: {
    type: [
      {
        combination: {
          type: [String],
          required: true,
        },
        probability: {
          type: Number,
          required: true,
        },
      },
    ],
    required: true,
  },
  dataId: { type: Types.ObjectId, required: true },
})

module.exports = model('CombinationsProbability', combinationsSchema)

auth.middleware.js
const jwt = require('jsonwebtoken')
const config = require('config')

module.exports = (req, res, next) => {
  if (req.method === 'OPTIONS') {
    return next()
  }

  try {
    const token = req.headers.authorization.split(' ')[1] // "Bearer TOKEN"

    if (!token) {
      return res.status(401).json({ message: 'Немає авторизації' })
    }

    const { exp } = jwt.decode(token)
    if (Date.now() >= exp * 1000) {
      return res
        .status(401)
        .json({ message: 'Сплив час існування токену', expirationIn: exp })
    }

    const decoded = jwt.verify(token, config.get('jwtSecret'))
    req.user = decoded
    next()
  } catch (e) {
    res.status(401).json({ message: 'Немає авторизації' })
  }
}

Default.json
{
  "port": 5000,
  "mongoUri": "mongodb+srv://Ostap-
Tadey:HdCwJfAhtuuOltPs@cluster0.kmbpxn4.mongodb.net/?retryWrites=true&w=majority",
  "passwordDb": "HdCwJfAhtuuOltPs",
  "jwtSecret": "test secret",
```

```
  "baseUrl": "http://localhost:5000"
}
```

Client/package.json

```
{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@emotion/react": "^11.11.0",
    "@emotion/styled": "^11.11.0",
    "@mui/icons-material": "^5.11.16",
    "@mui/material": "^5.13.2",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "materialize-css": "^1.0.0-rc.2",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.11.2",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "proxy": "http://localhost:5000",
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

Client/app.js

```
import React from 'react'
import 'materialize-css'
import { useRoutes } from './routes'
import { BrowserRouter } from 'react-router-dom'
import { useAuth } from './hooks/auth.hook'
import { AuthContext } from './context/AuthContext'
import { Navbar } from './components/Navbar'
import { DataIdContext } from './context/DataIdContext'
import { useDataId } from './hooks/dataId.hook'

function App() {
  const { token, login, logout, userId, tokenExpired, setUpTokenExpired } =
    useAuth()
  const isAuthenticated = !!token
  const routes = useRoutes(isAuthenticated, tokenExpired)
  const { dataId, setUpDataId } = useDataId()

  return (
    <DataIdContext.Provider value={{ dataId, setUpDataId }}>
```

```
      <AuthContext.Provider
        value={{
          token,
          login,
          logout,
          userId,
          isAuthenticated,
          tokenExpired,
          setUpTokenExpired,
        }}
      >
        <BrowserRouter>
          {isAuthenticated && !tokenExpired && <Navbar />}
          <div className="container">{routes}</div>
        </BrowserRouter>
      </AuthContext.Provider>
    </DataIdContext.Provider>
  )
}

export default App
client/index.js
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();

client/index.css
@import '~materialize-css/dist/css/materialize.min.css';
Client/routes.js
import { Route, Routes, Navigate } from 'react-router-dom'
import { FirstStageResultsPage } from './pages/FirstStageResultsPage'
import { InitialPage } from './pages/InitialDataPage'
import { AuthPage } from './pages/AuthPage'
import { SecondStagePage } from './pages/SecondStagePage'

export const useRoutes = (isAuthenticated, tokenExpired) => {
  if (isAuthenticated && !tokenExpired) {
    return (
      <Routes>
        <Route
          exact
          path="/create/:id"
          element={<FirstStageResultsPage />}
        ></Route>
        <Route exact path="/detail" element={<InitialPage />}></Route>
        <Route exact path="/auth" element={<Navigate to="/detail" />}></Route>
        <Route exact path="/second/:id" element={<SecondStagePage />}></Route>
        {/* <Route exact path="/detail" element={<Navigate to="/auth" />}></Route> */}
      </Routes>
    )
  }
  return (
    <Routes>
      <Route path="/" element={<Navigate to="/auth" />}></Route>
      <Route path="/auth" element={<AuthPage />}></Route>
      {tokenExpired && (
```

```
        <Route exact path="/*" element={<Navigate to="/auth" />}></Route>
      )}
    </Routes>
  )
}

Client/authPage.js
import React, { useContext, useEffect, useState } from 'react'
import { useHttp } from '../hooks/http.hook'
import { useMessage } from '../hooks/message.hook'
import { AuthContext } from '../context/AuthContext'
import {
  TextField,
  Button,
  Grid,
  Card,
  CardContent,
  Typography,
  CardActions,
} from '@mui/material'

export const AuthPage = () => {
  const auth = useContext(AuthContext)
  const message = useMessage()
  const [form, setForm] = useState({ email: '', password: '' })
  const { loading, request, error, clearError } = useHttp()

  useEffect(() => {
    message(error)
    clearError()
  }, [error, message, clearError])

  useEffect(() => {
    window.M.updateTextFields()
  }, [])

  const changeHandler = (event) => {
    setForm({ ...form, [event.target.name]: event.target.value })
  }

  const registerHandler = async () => {
    try {
      const data = await request('/api/auth/register', 'POST', { ...form })
    } catch (e) {}
  }

  const loginHandler = async () => {
    try {
      const data = await request('/api/auth/login', 'POST', { ...form })
      auth.login(data.token, data.userId)
    } catch (e) {}
  }

  return (
    <div
      style={{
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
        height: '100vh',
      }}
    >
      <Grid container justifyContent="center">
        <Grid item xs={12} sm={6} md={4}>
          <Card>
            <CardContent>
              <Typography variant="h5" component="div">
                Авторизація
              </Typography>
              <div>
```

```
                <TextField
                  fullWidth
                  label="Емейл"
                  id="email"
                  type="text"
                  name="email"
                  value={form.email}
                  onChange={changeHandler}
                  margin="normal"
                  variant="outlined"
                />
                <TextField
                  fullWidth
                  label="Пароль"
                  id="password"
                  type="password"
                  name="password"
                  value={form.password}
                  onChange={changeHandler}
                  margin="normal"
                  variant="outlined"
                />
              </div>
            </CardContent>
            <CardActions>
              <Button
                variant="contained"
                disabled={loading}
                onClick={loginHandler}
              >
                Увійти
              </Button>
              <Button
                variant="contained"
                onClick={registerHandler}
                disabled={loading}
              >
                Зареєструватися
              </Button>
            </CardActions>
          </Card>
        </Grid>
      </Grid>
    </div>
  )
}
```

Client/firstStageResultsPage.js

```
import React, { useContext, useEffect, useState } from 'react'
import { CombinationTable } from '../components/tables/CombinationsTable'
import { useHttp } from '../hooks/http.hook'
import { useParams } from 'react-router-dom'
import { AuthContext } from '../context/AuthContext'
import { DataIdContext } from '../context/DataIdContext'

export const FirstStageResultsPage = () => {
  const auth = useContext(AuthContext)

  const dataId = useParams().id
  const currentData = useContext(DataIdContext)
  currentData.setUpDataId(dataId)

  const { request } = useHttp()
  const [array, setArray] = useState([])
  const [inputArray, setInputArray] = useState([])
  const [matrixOfRelationships, setMatrixOfRelationships] = useState([])

  const handleGetRelationalData = async () => {
    try {
      const relationsData = await request(
```

```
      `/api/input/initial/relations/${dataId}`,
      'GET',
      null,
      { Authorization: `Bearer ${auth.token}` }
    )

    setMatrixOfRelationships([...relationsData.data.finalRes])

    return relationsData.data.relationsArray
  } catch (e) {}
}

const handleGetInitialData = async () => {
  try {
    const initialData = await request(
      `/api/input/initial/${dataId}`,
      'GET',
      null,
      { Authorization: `Bearer ${auth.token}` }
    )
    setInputArray(initialData.data.inputArray)
    const transformedInitialData = initialData.data.inputArray.map((row) => ({
      parameter: row.parameter,
      alternatives: row.alternatives,
      alternativesValues: row.alternativesValues,
    }))
    setArray(transformedInitialData)
  } catch (e) {}
}

useEffect(() => {
  handleGetRelationalData()
}, [])

useEffect(() => {
  handleGetInitialData()
}, [])

const handleTokenExpiration = async () => {
  try {
    const fetched = await request(
      '/api/token',
      'GET',
      null,
      {
        Authorization: `Bearer ${auth.token}`,
      },
      true
    )
    auth.setUpTokenExpired(false)
  } catch (e) {
    auth.setUpTokenExpired(true)
  }
}

useEffect(() => {
  handleTokenExpiration()
}, [])

return (
  <div>
    <h4>Ймовірності альтернатив разом з конфігураціями</h4>
    {matrixOfRelationships.length > 0 ? (
      <CombinationTable
        array={array}
        matrixOfRelationships={matrixOfRelationships}
      ></CombinationTable>
    ) : (
      <p1>You have to wait</p1>
    )}
```

```
    </div>
  )
}
```

Client/initialDataPage.js

```
import React, { useContext, useEffect, useState } from 'react'
import { InitialInputTable } from '../components/tables/InitialTable'
import { RelationsTable } from '../components/tables/RelationsTable'
import { useHttp } from '../hooks/http.hook'
import { Button } from '@mui/material'
import { DataIdContext } from '../context/DataIdContext'
import { AuthContext } from '../context/AuthContext'
import { EditTableContext } from '../context/EditTable.context'
import { useEditTable } from '../hooks/editTable.hook'

export const InitialPage = () => {
  const { token, setUpTokenExpired } = useContext(AuthContext)

  const {
    isModified,
    setUpIsDataModified,
    isSaved,
    setUpIsDataSaved,
    isRendered,
    setUpIsRelationsRendered,
  } = useEditTable(EditTableContext)

  const currentData = useContext(DataIdContext)

  const [array, setArray] = useState([])

  const { request } = useHttp()

  const handleTokenExpiration = async () => {
    try {
      const fetched = await request(
        '/api/token',
        'GET',
        null,
        {
          Authorization: `Bearer ${token}`,
        },
        true
      )
      setUpTokenExpired(false)
    } catch (e) {
      setUpTokenExpired(true)
    }
  }

  useEffect(() => {
    handleTokenExpiration()
  }, [])

  const handleGetData = async () => {
    try {
      let data
      if (currentData.dataId == undefined) {
        const fetched = await request('/api/input/initial', 'GET', null, {
          Authorization: `Bearer ${token}`,
        })
        data = fetched.data.pop()
        currentData.setUpDataId(data._id)
      } else {
        const fetched = await request(
          `/api/input/initial/${currentData.dataId}`,
          'GET',
          null,
          {
            Authorization: `Bearer ${token}`,
```

```
          }
        )
        data = fetched.data
      }
      const transformedData = data.inputArray.map((el) => ({
        parameter: el.parameter,
        alternatives: [...el.alternatives],
      }))
      setArray(transformedData)
    } catch (e) {}
  }

  const transformedArray = array.reduce(
    (result, { parameter, alternatives }) => {
      const parameterAlternatives = alternatives.map((alternative) => ({
        parameter,
        alternative,
      }))
      return [...result, ...parameterAlternatives]
    },
    []
  )

  return (
    <EditTableContext.Provider
      value={{
        isModified,
        setUpIsDataModified,
        isSaved,
        setUpIsDataSaved,
        isRendered,
        setUpIsRelationsRendered,
      }}
    >
      {/* <DataIdContext.Provider value={{ dataId, setUpDataId }}> */}
      <div>
        <h3>Введіть дані</h3>
        <InitialInputTable />
        <div style={{ margin: '16px' }}>
          {!isRendered && isSaved && (
            <Button
              variant="contained"
              onClick={() => {
                handleGetData()
              }}
            >
              Перерахувати таблицю взаємозв'язків
            </Button>
          )}
        </div>
        {!isModified && <h3>Матриця взаємозв'язків</h3>}
        {!isModified && (
          <RelationsTable
            array={transformedArray}
            dataId={currentData.dataId}
          />
        )}
      </div>
      {/* </DataIdContext.Provider> */}
    </EditTableContext.Provider>
  )
}
```

Client/secondStagePage.js

```
import { useContext, useEffect, useState } from 'react'
import { SecondStageInput } from '../components/tables/SecondStageInput'
import { useHttp } from '../hooks/http.hook'
import { AuthContext } from '../context/AuthContext'
import { useParams } from 'react-router-dom'
```

```
export const SecondStagePage = () => {
  const auth = useContext(AuthContext)

  const dataId = useParams().id

  const { request } = useHttp()

  const [combinations, setCombinations] = useState([])

  const [inputArray, setInputArray] = useState([])

  const handleGetInitialData = async () => {
    try {
      const initialData = await request(
        `/api/input/initial/${dataId}`,
        'GET',
        null,
        {
          Authorization: `Bearer ${auth.token}`,
        }
      )

      setInputArray(initialData.data.inputArray)
    } catch (e) {}
  }
  useEffect(() => {
    handleGetInitialData()
  }, [])

  const handleGetCombinationsData = async () => {
    try {
      const combinationsData = await request(
        `/api/input/initial/combinations/${dataId}`,
        'GET',
        null,
        { Authorization: `Bearer ${dataId}` }
      )

      setCombinations(combinationsData.data.combinationsProbability)
    } catch (e) {}
  }
  useEffect(() => {
    handleGetCombinationsData()
  }, [])

  const handleTokenExpiration = async () => {
    try {
      const fetched = await request(
        '/api/token',
        'GET',
        null,
        {
          Authorization: `Bearer ${auth.token}`,
        },
        true
      )

      auth.setUpTokenExpired(false)
    } catch (e) {
      auth.setUpTokenExpired(true)
    }
  }

  useEffect(() => {
    handleTokenExpiration()
  }, [])

  return (
    <div>
      <h3>Другий етап</h3>
```

```
      <SecondStageInput array={inputArray} combinations={combinations} />
    </div>
  )
}
```

Client/Navbar.js

```
import React, { useContext } from 'react'
import { NavLink, useNavigate } from 'react-router-dom'
import { AuthContext } from '../context/AuthContext'
import { DataIdContext } from '../context/DataIdContext'
import AppBar from '@mui/material/AppBar'
import Toolbar from '@mui/material/Toolbar'
import Typography from '@mui/material/Typography'
import Button from '@mui/material/Button'

export const Navbar = () => {
  const history = useNavigate()
  const auth = useContext(AuthContext)
  const logoutHandler = (event) => {
    event.preventDefault()
    auth.logout()
    history('/auth')
  }

  const currentData = useContext(DataIdContext)

  return (
    <AppBar position="static">
      <Toolbar>
        <Typography variant="h6" component="div" sx={{ flexGrow: 1 }}>
          <NavLink
            to="/detail"
            style={{ color: 'inherit', textDecoration: 'none' }}
          >
            Двоетапний метод морфологічного аналізу
          </NavLink>
        </Typography>
        <Button color="inherit" component={NavLink} to="/detail">
          Введення даних
        </Button>
        {currentData.dataId && (
          <Button
            color="inherit"
            component={NavLink}
            to={`/create/${currentData.dataId}`}
          >
            Оцінка
          </Button>
        )}
        {currentData.dataId && (
          <Button
            color="inherit"
            component={NavLink}
            to={`/second/${currentData.dataId}`}
          >
            Другий етап
          </Button>
        )}
        <Button color="inherit" onClick={logoutHandler}>
          Вийти
        </Button>
      </Toolbar>
    </AppBar>
  )
}
```

Client/initialTable.js

```
import React, { useContext, useState } from 'react'
import {
  Paper,
```

```
  Table,
  TableHead,
  TableRow,
  TableCell,
  TableBody,
  Button,
  TextField,
  Box,
  Typography,
  IconButton,
  InputBase,
} from '@mui/material'
import { useHttp } from '../../hooks/http.hook'
import { AuthContext } from '../../context/AuthContext'
import { DataIdContext } from '../../context/DataIdContext'
import { Delete as DeleteIcon, Add as AddIcon } from '@mui/icons-material'
import { EditTableContext } from '../../context/EditTable.context'

export const InitialInputTable = () => {
  const currentData = useContext(DataIdContext)

  const auth = useContext(AuthContext)
  const [rows, setRows] = useState([])

  const modifyTable = useContext(EditTableContext)

  const { request } = useHttp()

  const addRow = () => {
    setRows((previousRows) => [
      ...previousRows,
      { parameter: '', alternatives: [], alternativesValues: [] },
    ])
  }

  const deleteRow = (index) => {
    setRows((prevRows) => prevRows.filter((_, i) => i !== index))
  }

  const deleteAlternative = (rowIndex, alternativeIndex) => {
    const updatedRows = [...rows]
    updatedRows[rowIndex].alternatives.splice(alternativeIndex, 1)
    updatedRows[rowIndex].alternativesValues.splice(alternativeIndex, 1)
    modifyTable.setUpIsDataModified(true)
    modifyTable.setUpIsRelationsRendered(false)

    setRows(updatedRows)
  }

  const handleParameterChange = (index, event) => {
    const updatedRows = [...rows]
    updatedRows[index].parameter = event.target.value
    modifyTable.setUpIsDataModified(true)
    modifyTable.setUpIsRelationsRendered(false)

    setRows(updatedRows)
  }

  const handleAlternativeChange = (rowIndex, alternativeIndex, event) => {
    const updatedRows = [...rows]
    updatedRows[rowIndex].alternatives[alternativeIndex] = event.target.value
    modifyTable.setUpIsDataModified(true)
    modifyTable.setUpIsRelationsRendered(false)

    setRows(updatedRows)
  }

  const [sumExceededRows, setSumExceededRows] = useState([])
  const [parameterSums, setParameterSums] = useState({})
```

```
const handleValueChange = (rowIndex, alternativeIndex, event) => {
  const updatedRows = [...rows]
  const value = parseFloat(event.target.value)

  // Calculate the sum for the current parameter
  const currentParameter = updatedRows[rowIndex].parameter
  const currentSum = updatedRows[rowIndex].alternativesValues.reduce(
    (sum, val, index) => {
      if (index !== alternativeIndex) {
        return sum + parseFloat(val)
      }
      return sum + value
    },
    0
  )

  // Update the sum and check if it exceeds 1
  const sumExceedsLimit = currentSum > 1
  setParameterSums((prevSums) => ({
    ...prevSums,
    [currentParameter]: currentSum,
  }))

  // Update the list of rows where sum exceeds 1
  setSumExceededRows((prevRows) => {
    if (sumExceedsLimit && !prevRows.includes(rowIndex)) {
      return [...prevRows, rowIndex]
    }
    if (!sumExceedsLimit && prevRows.includes(rowIndex)) {
      return prevRows.filter((row) => row !== rowIndex)
    }
    return prevRows
  })

  // Restrict input to numbers between -1 and 1 if sum exceeds 1
  if (!sumExceedsLimit || (value >= 0 && value <= 1)) {
    updatedRows[rowIndex].alternativesValues[alternativeIndex] = value
    setRows(updatedRows)
    modifyTable.setUpIsDataModified(true)
    modifyTable.setUpIsRelationsRendered(false)
  }
}

const handleSaveData = async () => {
  if (rows.length < 2) {
    alert('At least 2 parameters are required')
    return
  }

  for (let i = 0; i < rows.length; i++) {
    const row = rows[i]
    if (row.parameter.trim() === '') {
      alert('Not all fields are filled with data')
      return
    }

    if (row.alternatives.length < 2) {
      alert(`At least two alternatives are required for "${row.parameter}"`)
      return
    }

    for (let j = 0; j < row.alternatives.length; j++) {
      if (
        row.alternatives[j].trim() === '' ||
        row.alternativesValues[j] === ''
      ) {
        alert('Not all fields are filled with data')
        return
      }
    }
```

```
  }

  try {
    const data = await request(
      '/api/input/initial',
      'POST',
      { rows },
      { Authorization: `Bearer ${auth.token}` }
    )
    currentData.setUpDataId(data.dataId)
    modifyTable.setUpIsDataSaved(true)
  } catch (e) {

  }
}

return (
  <>
    <Paper>
      <Table>
        <TableHead>
          <TableRow>
            <TableCell align="center">Parameter</TableCell>
            <TableCell align="center">Alternatives</TableCell>
            <TableCell align="center">Values</TableCell>
            <TableCell></TableCell>
          </TableRow>
        </TableHead>
        <TableBody>
          {rows.map((row, index) => (
            <TableRow key={index}>
              <TableCell align="center">
                <InputBase
                  value={row.parameter}
                  onChange={(event) => handleParameterChange(index, event)}
                  inputProps={{ style: { textAlign: 'center' } }}
                />
              </TableCell>
              <TableCell align="center">
                <Box display="flex" flexDirection="column">
                  {row.alternatives.map((alternative, alternativeIndex) => (
                    <Box
                      key={`alternative-${index}-${alternativeIndex}`}
                      marginBottom={1}
                    >
                      <InputBase
                        value={alternative}
                        onChange={(event) =>
                          handleAlternativeChange(
                            index,
                            alternativeIndex,
                            event
                          )
                        }
                        inputProps={{ style: { textAlign: 'center' } }}
                      />
                      <IconButton
                        onClick={() => {
                          deleteAlternative(index, alternativeIndex)
                          modifyTable.setUpIsDataModified(true)
                          modifyTable.setUpIsRelationsRendered(false)
                        }}
                        aria-label="Delete Alternative"
                      >
                        <DeleteIcon />
                      </IconButton>
                    </Box>
                  ))}
                </Box>
                <Button
```

```
            variant="outlined"
            onClick={() => {
              const updatedRows = [...rows]
              updatedRows[index].alternatives.push('')
              updatedRows[index].alternativesValues.push('')
              setRows(updatedRows)
              modifyTable.setUpIsDataModified(true)
              modifyTable.setUpIsRelationsRendered(false)
            }}
            startIcon={<AddIcon />}
            style={{
              textAlign: 'center',
              border: 'none',
              outline: 'none',
            }}
          >
            ДОДАТИ АЛЬТЕРНАТИВУ
          </Button>
        </TableCell>
        <TableCell>
          <Box display="flex" flexDirection="column">
            {row.alternatives.map((alternative, alternativeIndex) => (
              <InputBase
                key={`value-${index}-${alternativeIndex}`}
                value={row.alternativesValues[alternativeIndex]}
                onChange={(event) =>
                  handleValueChange(index, alternativeIndex, event)
                }
                style={{ marginBottom: '8px' }}
                type="number"
                inputProps={{
                  min: 0,
                  max: 1,
                  step: 0.01,
                  style: { textAlign: 'center' },
                }}
                error={sumExceededRows.includes(index)}
                helperText={
                  sumExceededRows.includes(index)
                    ? 'Sum exceeded 1'
                    : null
                }
              />
            ))}
          </Box>
        </TableCell>
        <TableCell>
          <IconButton
            onClick={() => {
              deleteRow(index)
              modifyTable.setUpIsDataModified(true)
              modifyTable.setUpIsRelationsRendered(false)
            }}
            aria-label="Delete Row"
          >
            <DeleteIcon />
          </IconButton>
        </TableCell>
      </TableRow>
    ))}
  </TableBody>
</Table>
<Button
  variant="contained"
  onClick={() => {
    addRow()
    modifyTable.setUpIsDataModified(true)
    modifyTable.setUpIsRelationsRendered(false)
  }}
  startIcon={<AddIcon />}
```

```
          style={{ margin: '16px' }}
        >
          ДОДАТИ ПАРАМЕТР
        </Button>
        <Button
          variant="contained"
          onClick={() => {
            handleSaveData()
            modifyTable.setUpIsDataModified(false)
            modifyTable.setUpIsRelationsRendered(false)
          }}
          style={{ margin: '16px' }}
        >
          ЗБЕРЕГТИ
        </Button>
      </Paper>
    </>
  )
}
```

Client/relationsTable.js

```
import React, { useCallback, useContext, useEffect, useState } from 'react'
import {
  TableContainer,
  Table,
  TableHead,
  TableRow,
  TableCell,
  TableBody,
  styled,
  Button,
  Paper,
  InputBase,
} from '@mui/material'
import { useHttp } from '../../hooks/http.hook'
import { EditTableContext } from '../../context/EditTable.context'

const useStyles = styled({
  inputCell: {
    width: '20px', // Adjust the width to your desired size
    padding: '4px', // Adjust the padding to control the distance between cells
  },
})

export const RelationsTable = ({ array, dataId }) => {
  const { request } = useHttp()
  const classes = useStyles()

  const [relations, setRelations] = useState()

  const modifyTable = useContext(EditTableContext)

  useEffect(() => {
    if (array && array.length > 0) {
      const initialRelations = array.map(() => new Array(array.length).fill(0))
      setRelations(initialRelations)
    }
  }, [array])

  const handleRelationChange = useCallback(
    (rowElement, colElement, rowIndex, colIndex, event) => {
      modifyTable.setUpIsRelationsRendered(true)
      const value = event.target.value
      const parsedValue = parseFloat(value)

      if (!Number.isNaN(parsedValue) && parsedValue >= -1 && parsedValue <= 1) {
        const updatedRelations = [...relations]

        if (rowElement.parameter === colElement.parameter) {
          updatedRelations[rowIndex][colIndex] = 0
```

```
        return
      }

      updatedRelations[rowIndex][colIndex] = parsedValue
      updatedRelations[colIndex][rowIndex] = parsedValue // Update the mirrored cell

      setRelations(updatedRelations)
    }
  },
  [relations]
)

const getRelationalValue = (rowElement, colElement, rowIndex, colIndex) => {
  if (
    !relations ||
    rowIndex >= relations.length ||
    colIndex >= relations[rowIndex].length
  ) {
    return 0
  }
  if (rowElement.parameter === colElement.parameter) {
    return 0
  }
  return relations[rowIndex][colIndex]
}

const renderTableHead = () => (
  <TableHead>
    <TableRow>
      <TableCell></TableCell>
      {array.map((element, colIndex) => (
        <TableCell key={colIndex}>{element.alternative}</TableCell>
      ))}
    </TableRow>
  </TableHead>
)

const handlePostData = async () => {
  try {
    const altValues = []
    array.forEach((rowElement, rowIndex) => {
      const baseAltName = rowElement.alternative
      array.forEach((colElement, colIndex) => {
        const relatedAlt = colElement.alternative
        const value = relations[rowIndex][colIndex]
        altValues.push({ baseAltName, relatedAlt, value })
      })
    })

    const indexes = []

    altValues.forEach((relation, relationIdx) => {
      altValues.reduce((_, compareRelation, compareRelationIdx) => {
        return relation.baseAltName === compareRelation.relatedAlt &&
          relation.relatedAlt === compareRelation.baseAltName &&
          relationIdx !== compareRelationIdx &&
          relationIdx < compareRelationIdx
          ? indexes.push(compareRelationIdx)
          : true
      }, relationIdx)
    })

    const finalRes = altValues.filter((value, idx) => {
      return indexes.findIndex((chosenIdx) => chosenIdx === idx) === -1
    })

    const data = await request('/api/input/initial/relations', 'POST', {
      finalRes,
      dataId,
    })
```

```
    window.open(`/create/${data.input.dataId}`, '_blank', 'noreferrer')
  } catch (e) {}
}

const renderTableBody = () => (
  <TableBody>
    {array.map((rowElement, rowIndex) => (
      <TableRow key={rowIndex}>
        <TableCell>{rowElement.alternative}</TableCell>
        {array.map((colElement, colIndex) => (
          <TableCell key={colIndex}>
            <InputBase
              value={getRelationalValue(
                rowElement,
                colElement,
                rowIndex,
                colIndex
              )}
              onChange={(event) =>
                handleRelationChange(
                  rowElement,
                  colElement,
                  rowIndex,
                  colIndex,
                  event
                )
              }
              inputProps={{
                step: 0.01,
                min: -1,
                max: 1,
                type: 'number',
                pattern: '^(-1)?([01](\\.\\d{0,2})?)?$',
                style: {
                  textAlign: 'center',
                  border: 'none',
                  outline: 'none',
                },
              }}
            />
          </TableCell>
        ))}
      </TableRow>
    ))}
  </TableBody>
)

return (
  <Paper>
    <TableContainer>
      <Table>
        {renderTableHead()}
        {renderTableBody()}
      </Table>
      <div style={{ margin: '20px' }}>
        {modifyTable.isRendered && modifyTable.isSaved && (
          <Button
            variant="contained"
            onClick={() => {
              handlePostData()
            }}
          >
            ОЦІНКИ АЛЬТЕРНАТИВ
          </Button>
        )}
      </div>
    </TableContainer>
  </Paper>
)
```

```
}

Client/combinationsTable.js
import React, { useState } from 'react'
import {
  TableContainer,
  Table,
  TableHead,
  TableRow,
  TableCell,
  TableBody,
  Paper,
  Button,
} from '@mui/material'
import { useHttp } from '../../hooks/http.hook'
import { useParams } from 'react-router-dom'

const generateCombinations = (arrays) => {
  const combinations = []
  const helper = (arr, i, currentCombination) => {
    if (i === arr.length) {
      combinations.push(currentCombination)
    } else {
      for (let j = 0; j < arr[i].length; j++) {
        helper(arr, i + 1, [...currentCombination, arr[i][j]])
      }
    }
  }

  helper(arrays, 0, [])
  return combinations
}

export const CombinationTable = ({ array, matrixOfRelationships }) => {
  const dataId = useParams().id

  const combinations = generateCombinations(
    array.map((item) => item.alternatives)
  )

  const calculateMatrixValue = (combination) => {
    let result = 1
    if (matrixOfRelationships) {
      for (let i = 0; i < combination.length - 1; i++) {
        let elementPairsValues
        for (let j = i + 1; j <= combination.length - 1; j++) {
          const firstAltName = combination[i]
          const secondAltName = combination[j]

          elementPairsValues = matrixOfRelationships.reduce(
            (acc, curr) =>
              Object.values(curr).includes(firstAltName) &&
              Object.values(curr).includes(secondAltName)
                ? acc * (curr.value + 1)
                : acc + 0,
            1
          )
        }
        result *= elementPairsValues
      }

      return result.toFixed(20)
    }
  }

  const calculateCombinationValue = (combination) => {
    let result = 1
    for (let i = 0; i < combination.length; i++) {
      const parameterIndex = array[i].alternatives.indexOf(combination[i])
      const alternativeValue = array[i].alternativesValues[parameterIndex]
```

```
    result *= alternativeValue
  }
  return result.toFixed(20)
}

const calculateTotalValue = (combination) => {
  const matrixValue = calculateMatrixValue(combination)
  const combinationValue = calculateCombinationValue(combination)
  return (matrixValue * combinationValue).toFixed(20)
}

const calculateNormalizedValue = (value) => {
  const normalizedValue = value / sumOfValues
  return normalizedValue.toFixed(20)
}

const totalValues = combinations.map((combination) =>
  calculateTotalValue(combination)
)
const sumOfValues = totalValues.reduce((acc, curr) => acc + +curr, 0)

const [savedData, setSavedData] = useState([])

const { request } = useHttp()

const handlePostCombinationProbability = async (combinationData) => {
  try {
    const dataToPost = combinationData.map((row) => {
      return {
        combination: row.combination,
        probability: row.normalizedValue,
      }
    })

    const data = await request('/api/input/initial/combinations', 'POST', {
      combinationsProbability: dataToPost,
      dataId: dataId,
    })
  } catch (e) {}
}

const handleSaveData = () => {
  const newData = combinations.map((combination, index) => ({
    combination: combination,
    matrixValue: calculateMatrixValue(combination),
    combinationValue: calculateCombinationValue(combination),
    totalValue: calculateTotalValue(combination),
    normalizedValue: calculateNormalizedValue(
      calculateTotalValue(combination)
    ),
  }))

  setSavedData(newData)
}

const transformedArray = array.reduce(
  (result, { parameter, alternatives }) => {
    const parameterAlternatives = alternatives.map((alternative) => ({
      parameter,
      alternative,
    }))
    return [...result, ...parameterAlternatives]
  },
  []
)

const countProbability = () => {
  const transformedData = transformedArray.map((element) => {
    const probability = savedData.reduce((acc, curr) => {
      if (curr.combination.indexOf(element.alternative) !== -1) {
```

```
        return acc + +curr.normalizedValue
      }
      return acc + 0
    }, 0)
    return { element, probability }
  })
  return transformedData
}
const finalArray = countProbability()

const parameterMap = new Map()

// Group the data by parameter
finalArray.forEach((item) => {
  const { parameter, alternative } = item.element
  const probability = item.probability
  if (!parameterMap.has(parameter)) {
    parameterMap.set(parameter, [])
  }
  parameterMap.get(parameter).push({ alternative, probability })
})

return (
  <div>
    <Paper>
      <TableContainer style={{ maxHeight: '400px' }}>
        <Table stickyHeader>
          <TableHead>
            <TableRow>
              {array.map((param) => (
                <TableCell key={param.parameter}>{param.parameter}</TableCell>
              ))}
              <TableCell>C</TableCell>
              <TableCell>p</TableCell>
              <TableCell>pC</TableCell>
              <TableCell>P</TableCell>
            </TableRow>
          </TableHead>
          <TableBody>
            {/* Render table content only when matrixOfRelationships has a value */}
            {matrixOfRelationships && matrixOfRelationships.length > 0 ? (
              combinations.map((combination, index) => (
                <TableRow key={index}>
                  {/* Render table cells for combination values */}
                  {combination.map((alternative, altIndex) => (
                    <TableCell key={altIndex}>{alternative}</TableCell>
                  ))}
                  <TableCell>{calculateMatrixValue(combination)}</TableCell>
                  <TableCell>
                    {calculateCombinationValue(combination)}
                  </TableCell>
                  <TableCell>{calculateTotalValue(combination)}</TableCell>
                  <TableCell>
                    {calculateNormalizedValue(
                      calculateTotalValue(combination)
                    )}
                  </TableCell>
                </TableRow>
              ))
            ) : (
              // Render a placeholder row when matrixOfRelationships is undefined
              <TableRow>
                <TableCell colSpan={array.length + 4}>Loading...</TableCell>
              </TableRow>
            )}
          </TableBody>
        </Table>
      </TableContainer>
    </Paper>
    <div style={{ margin: '16px', marginBottom: 0 }}>
```

```
        <Button onClick={handleSaveData}>ОБЧИСЛИТИ ОЦІНКИ</Button>
      </div>
      <Button></Button>

      <div>
        <h4>Оцінки альтернатив</h4>
        <TableContainer component={Paper} style={{ marginBottom: '20px' }}>
          <Table>
            <TableHead>
              <TableRow>
                <TableCell>Parameter</TableCell>
                <TableCell>Alternative</TableCell>
                <TableCell>Probability</TableCell>
              </TableRow>
            </TableHead>
            <TableBody>
              {Array.from(parameterMap, ([parameter, alternatives]) => (
                <React.Fragment key={parameter}>
                  <TableRow>
                    <TableCell rowSpan={alternatives.length}>
                      {parameter}
                    </TableCell>
                    <TableCell>{alternatives[0].alternative}</TableCell>
                    <TableCell>{alternatives[0].probability}</TableCell>
                  </TableRow>
                  {alternatives.slice(1).map((alt, index) => (
                    <TableRow key={index}>
                      <TableCell>{alt.alternative}</TableCell>
                      <TableCell>{alt.probability}</TableCell>
                    </TableRow>
                  ))}
                </React.Fragment>
              ))}
            </TableBody>
          </Table>
          <div style={{ margin: '16px' }}>
            {savedData.length > 0 && (
              <Button
                onClick={() => {
                  window.open(`/second/${dataId}`, '_blank', 'noreferrer')
                  handlePostCombinationProbability(savedData)
                }}
              >
                ДРУГИЙ ЕТАП
              </Button>
            )}
          </div>
        </TableContainer>
      </div>
    </div>
  )
}

Client/SecondStageInput.js
import React, { useState } from 'react'
import {
  Table,
  TableBody,
  TableCell,
  TableContainer,
  TableHead,
  TableRow,
  Paper,
  TextField,
  Button,
  IconButton,
  InputBase,
} from '@mui/material'

import {
```

```
  DisabledByDefault as DisableIcon,
  Add as AddIcon,
  DeleteOutlineRounded as DeleteIcon,
} from '@mui/icons-material'

export const SecondStageInput = ({ array, combinations }) => {
  const [groups, setGroups] = useState([])
  const [newGroupName, setNewGroupName] = useState('')

  const transformedArray = array.flatMap((element) => element.alternatives)

  const addGroup = () => {
    const newGroup = {
      name: newGroupName,
      columns: ['Альтернатива'],
      columnData: [],
    }
    setGroups([...groups, newGroup])
    setNewGroupName('')
  }

  const addColumn = (groupIndex) => {
    const updatedGroups = [...groups]
    updatedGroups[groupIndex].columns.push('Нова Альтернатива')
    setGroups(updatedGroups)
  }

  const updateColumnName = (groupIndex, columnIndex, newName) => {
    const updatedGroups = [...groups]
    updatedGroups[groupIndex].columns[columnIndex] = newName
    setGroups(updatedGroups)
  }

  const updateColumnValue = (groupIndex, columnName, altIndex, value) => {
    const updatedGroups = [...groups]
    const group = updatedGroups[groupIndex]
    let columnData = group.columnData.find(
      (column) => column.columnName === columnName
    )

    if (!columnData) {
      columnData = {
        columnName: columnName,
        values: {},
      }
      group.columnData.push(columnData)
    }

    const alternative = transformedArray[altIndex]
    columnData.values[alternative] = value

    setGroups(updatedGroups)
  }

  const sumArray = (array) => {
    const newArray = []
    array.forEach((sub) => {
      sub.forEach((num, index) => {
        if (newArray[index]) {
          newArray[index] += num
        } else {
          newArray[index] = num
        }
      })
    })
    return newArray
  }

  const calculateNormalized = (R) => {
    const rColumnsValues = R.map((column) => column.columnValues)
```

```
  const sumValuesArray = sumArray(rColumnsValues)

  return R.map((data, dataIdx) => {
    return {
      columnName: data.columnName,
      columnNormalizedValues: data.columnValues.map(
        (value) => +(value / sumValuesArray[dataIdx]).toFixed(20)
      ),
    }
  })
}

const calculateProbabilityProduct = (normalizedData) => {
  return normalizedData.map((normalizedEl) => {
    return {
      columnName: normalizedEl.columnName,
      columnProductValues: normalizedEl.columnNormalizedValues.map(
        (value, valueIdx) => value * combinations[valueIdx].probability
      ),
    }
  })
}

const GroupTable = ({
  groupName,
  columns,
  combinations,
  columnsData,
  groupIndex,
}) => {
  const alternativesParameters = array.map((item) => item.parameter)

  const alternativesIntersectionValues = columnsData.map((column) => {
    return { columnName: column.columnName, values: column.values }
  })

  let R = []
  if (alternativesIntersectionValues.length > 0) {
    R = alternativesIntersectionValues.map((columnObject) => {
      let values = []
      const column = columnObject.values
      for (let i = 0; i <= combinations.length - 1; i++) {
        let combination = combinations[i].combination
        const product = combination.reduce((acc, curr) => {
          const idxInColumn = Object.keys(column).indexOf(curr)
          let valueInColumn = Object.values(column)[idxInColumn]
          valueInColumn++
          return acc * valueInColumn
        }, 1)
        values.push(+product.toFixed(20))
      }

      return { columnName: columnObject.columnName, columnValues: values }
    })
  }

  const normalizedValues = calculateNormalized(R)
  const probabilityProduct = calculateProbabilityProduct(normalizedValues)
  const sumRValues = probabilityProduct.map((column) =>
    column.columnProductValues.reduce((acc, curr) => acc + curr, 0)
  )

  return (
    <div>
      <TableContainer component={Paper}>
        <Table>
          <TableHead>
            <TableRow>
              <TableCell>Ймовірність</TableCell>
              {alternativesParameters.map((name, nameIdx) => (
```

```
            <TableCell key={nameIdx} align="left">
              {name}
            </TableCell>
          ))}
          {R &&
            columns.map((_, columnIndex) => (
              <TableCell key={columnIndex} align="left">
                {`R\`${columnIndex + 1}`}
              </TableCell>
            ))}
          {normalizedValues &&
            normalizedValues.map((_, columnIndex) => (
              <TableCell key={columnIndex} align="left">
                {`R${columnIndex + 1}`}
              </TableCell>
            ))}
          {probabilityProduct &&
            probabilityProduct.map((_, columnIndex) => (
              <TableCell key={columnIndex} align="left">
                {`R${columnIndex + 1}P`}
              </TableCell>
            ))}
        </TableRow>
      </TableHead>
      <TableBody>
        {combinations.map((combination, combinationIndex) => (
          <TableRow key={combinationIndex}>
            <TableCell>{combination.probability}</TableCell>
            {combination.combination.map((alternative, columnIndex) => (
              <TableCell key={columnIndex} align="left">
                {alternative}
              </TableCell>
            ))}

            {R &&
              columnsData.map((_, columnIdx) => (
                <TableCell key={columnIdx}>
                  {R[columnIdx].columnValues[combinationIndex]}
                </TableCell>
              ))}
            {columnsData &&
              columnsData.map((_, columnIdx) => (
                <TableCell key={columnIdx}>
                  {
                    normalizedValues[columnIdx].columnNormalizedValues[
                      combinationIndex
                    ]
                  }
                </TableCell>
              ))}
            {columnsData &&
              columnsData.map((_, columnIdx) => (
                <TableCell key={columnIdx}>
                  {
                    probabilityProduct[columnIdx].columnProductValues[
                      combinationIndex
                    ]
                  }
                </TableCell>
              ))}
          </TableRow>
        ))}
      </TableBody>
    </Table>
  </TableContainer>
  <div style={{ marginBottom: '20px' }}>
    <h4>Очікуваний результат</h4>
    <h5>{`${groupName}`}</h5>
    <div>
      <TableContainer component={Paper}>
```

```
            <Table>
              <TableHead>
                <TableRow>
                  <TableCell>Альтернатива</TableCell>
                  <TableCell>Ймовірність</TableCell>
                </TableRow>
              </TableHead>
              <TableBody>
                {columns.map((column, columnIndex) => (
                  <TableRow key={columnIndex}>
                    <TableCell>{column}</TableCell>
                    <TableCell>{sumRValues[columnIndex]}</TableCell>
                  </TableRow>
                ))}
              </TableBody>
            </Table>
          </TableContainer>
        </div>
      </div>
    </div>
  )
}

const removeGroup = (groupIndex) => {
  const updatedGroups = [...groups]
  updatedGroups.splice(groupIndex, 1)
  setGroups(updatedGroups)
}

const removeColumn = (groupIndex, columnIndex) => {
  const updatedGroups = [...groups]
  updatedGroups[groupIndex].columns.splice(columnIndex, 1)
  updatedGroups[groupIndex].columnData.splice(columnIndex, 1)
  setGroups(updatedGroups)
}

return (
  <div>
    <div style={{ margin: '16px', marginLeft: 0 }}>
      <TextField
        label="Назва параметру"
        value={newGroupName}
        onChange={(e) => setNewGroupName(e.target.value)}
        inputProps={{ style: { textAlign: 'center' } }}
      />
      <Button
        variant="contained"
        onClick={addGroup}
        style={{ margin: '16px', marginTop: '8px' }}
      >
        Добавити Параметр
      </Button>
    </div>
    <TableContainer component={Paper}>
      <Table>
        <TableHead>
          <TableRow>
            <TableCell>Альтернативи</TableCell>
            {groups.map((group, groupIndex) => (
              <React.Fragment key={group.name}>
                <TableCell align="right" colSpan={group.columns.length}>
                  {group.name}
                </TableCell>

                <TableCell align="right">
                  <IconButton
                    onClick={() => removeGroup(groupIndex)}
                    size="small"
                  >
                    <DeleteIcon />
```

```
            </IconButton>
          </TableCell>
          <IconButton
            onClick={() => addColumn(groupIndex)}
            size="small"
          >
            <AddIcon />
          </IconButton>
        </React.Fragment>
      ))}
    </TableRow>
    <TableRow>
      <TableCell />
      {groups.map((group, groupIndex) => (
        <React.Fragment key={group.name}>
          {group.columns.map((column, columnIndex) => (
            <TableCell key={columnIndex} align="right">
              <InputBase
                value={column}
                variant="outlined"
                size="small"
                onChange={(e) =>
                  updateColumnName(
                    groupIndex,
                    columnIndex,
                    e.target.value
                  )
                }
                inputProps={{ style: { textAlign: 'center' } }}
              />
              <IconButton
                onClick={() => removeColumn(groupIndex, columnIndex)}
                size="small"
              >
                <DisableIcon />
              </IconButton>
            </TableCell>
          ))}
          {groupIndex < groups.length - 1 && <TableCell> </TableCell>}
        </React.Fragment>
      ))}
    </TableRow>
  </TableHead>

  <TableBody>
    {transformedArray.map((alternative, alternativeIndex) => (
      <TableRow key={alternativeIndex}>
        <TableCell>{alternative}</TableCell>
        {groups.map((group, groupIndex) => (
          <React.Fragment key={group.name}>
            {group.columns.map((currentColumnName, columnIndex) => (
              <TableCell key={columnIndex} align="right">
                <InputBase
                  type="number"
                  variant="outlined"
                  size="small"
                  inputProps={{
                    step: 0.01,
                    min: -1,
                    max: 1,
                    style: { width: '50px' },
                  }}
                  onChange={(e) =>
                    updateColumnValue(
                      groupIndex,
                      currentColumnName,
                      alternativeIndex,
                      e.target.value
                    )
                  }
```

```
                        InputLabelProps={{
                          shrink: true,
                        }}
                      />
                    </TableCell>
                  ))}
                  {groupIndex < groups.length - 1 && <TableCell> </TableCell>}
                </React.Fragment>
              ))}
            </TableRow>
          ))}
        </TableBody>
      </Table>
    </TableContainer>
    {groups.map((group, groupIndex) => (
      <div key={groupIndex}>
        <div>
          <h3>Таблиця конфігурацій</h3> <h4>{group.name}</h4>
        </div>
        <GroupTable
          groupName={group.name}
          columns={group.columns}
          combinations={combinations}
          columnsData={group.columnData}
          groupIndex={groupIndex}
        />
      </div>
    ))}
  </div>
 )
}

Client/auth.hook.js
import { useCallback, useEffect, useState } from 'react'

const storageName = 'userData'

export const useAuth = () => {
  const [token, setToken] = useState(null)
  const [userId, setUserId] = useState(null)
  const [tokenExpired, setTokenExpired] = useState(false)

  const setUpTokenExpired = useCallback((value) => {
    setTokenExpired(value)
  }, [])

  const login = useCallback((jwtToken, id) => {
    setToken(jwtToken)
    setUserId(id)
    localStorage.setItem(
      storageName,
      JSON.stringify({ userId: id, token: jwtToken })
    )
    setTokenExpired(false)
  }, [])

  const logout = useCallback(() => {
    setToken(null)
    setUserId(null)
    localStorage.removeItem(storageName)
  }, [])

  useEffect(() => {
    const data = JSON.parse(localStorage.getItem(storageName))
    if (data && data.token) {
      login(data.token, data.id)
    }
  }, [login])

  return { login, logout, token, userId, tokenExpired, setUpTokenExpired }
```

```
}

Client/dataId.hook.js
import { useCallback, useState } from 'react'

export const useDataId = () => {
  const [dataId, setDataId] = useState(null)

  const setUpDataId = useCallback((id) => {
    setDataId(id)
  }, [])

  return { dataId, setUpDataId }
}

Client/editTable.hook.js
import { useCallback, useState } from 'react'

export const useDataId = () => {
  const [dataId, setDataId] = useState(null)

  const setUpDataId = useCallback((id) => {
    setDataId(id)
  }, [])

  return { dataId, setUpDataId }
}
http.hook.js
import { useCallback, useState } from 'react'

export const useHttp = () => {
  const [loading, setLoading] = useState(false)
  const [error, setError] = useState(null)

  const request = useCallback(
    async (
      url,
      method = 'GET',
      body = null,
      headers = {},
      returnCode = false
    ) => {
      setLoading(true)
      try {
        if (body) {
          body = JSON.stringify(body)
          headers['Content-Type'] = 'application/json'
        }
        const response = await fetch(url, { method, body, headers })
        const data = await response.json()
        if (!response.ok) {
          throw new Error(data.message || 'Something went wrong')
        }
        setLoading(false)
        if (returnCode) {
          return {
            data,
            code: response.status,
          }
        }
        return data
      } catch (e) {
        setLoading(false)
        setError(e.message)
        throw e
      }
    },
    []
  )
  const clearError = useCallback(() => {
```

```
    setError(null)
  }, [])
  return { loading, request, error, clearError }
}

Client/message.hook.js
import { useCallback } from 'react'

export const useMessage = () => {
  return useCallback((text) => {
    if (window.M && text) {
      window.M.toast({ html: text })
    }
  }, [])
}

Client/authContext.js
import { createContext } from 'react'

function noop() {}

export const AuthContext = createContext({
  token: null,
  userId: null,
  login: noop(),
  logout: noop(),
  isAuthenticated: false,
  tokenExpired: false,
  setUpTokenExpired: noop(),
})


Client/DataIdContext.js
import { createContext } from 'react'

function noop() {}

export const AuthContext = createContext({
  token: null,
  userId: null,
  login: noop(),
  logout: noop(),
  isAuthenticated: false,
  tokenExpired: false,
  setUpTokenExpired: noop(),
})

Client/editable.context.js
import { createContext } from 'react'

function noop() {}

export const EditTableContext = createContext({
  isModified: true,
  setUpIsDataModified: noop(),
  isSaved: true,
  setUpIsDataSaved: noop(),
  isRendered: false,
  setUpIsRelationsRendered: noop(),
})
```

ДОДАТОК Б. Презентація

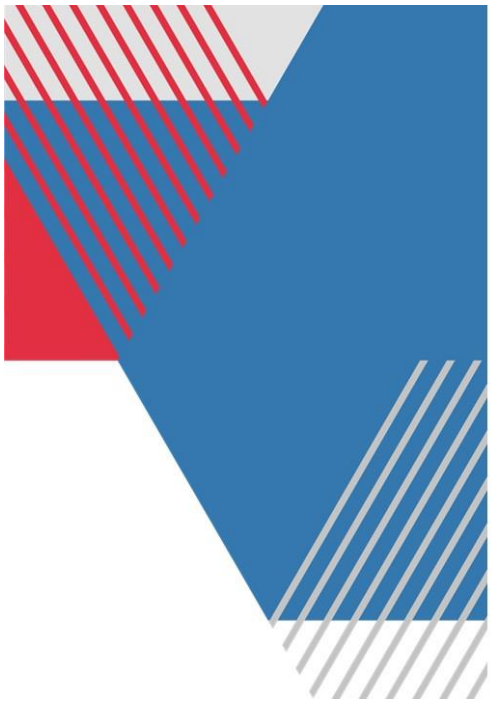
Середовище розв’язання задач сценарного аналізу на основі модифікованого методу морфологічного аналізу

Виконав: Демчишин Остап-Тадей Назарович

Керівник: Савченко Ілля Олександрович

Постановка задачі

- ✓ Розглянути модифікований метод морфологічного аналізу як інструмент для розв’язання задач сценарного типу
- ✓ Створити архітектуру програмного забезпечення, підібрати інструменти і бібліотеки які необхідні для розробки та розглянути механізми взаємодії усіх модулів
- ✓ Розробити програмний продукт
- ✓ Провести дослідження з використання веб-додатку, використовуючи тестову сценарну задачу



Актуальність

У процесі сценарного аналізу розглядаються різні версії розвитку подій чи систем, які залежать від певних факторів; виявляються точки, в яких відкриваються альтернативні шляхи до бажаних цілей та виробляються можливі способи їхнього досягнення, чи вирішення проблем, які можуть стати на шляху досягнення цілі

Розв'язання задач сценарного аналізу може бути використане для управління системами різної природи або ж формулювання та прийняття рішень у рамках цих систем.

Приклад сфер використання:

- економічна
- безпекова
- дослідницька

Об'єкт дослідження

Задачі сценарного аналізу

Предмет дослідження

Середовище розв'язання задач сценарного аналізу використовуючи двоетапний метод морфологічного аналізу

Мета дослідження

Розробка програмного продукту у вигляді середовища для розв'язання задач сценарного аналізу, яке б дозволяло здійснювати усі етапи модифікованого методу морфологічного аналізу, отримувати результати у вигляді оцінок та здійснювати аналіз

Морфологічний аналіз

Метод морфологічного аналізу є потужним інструментом в наукових дослідженнях для систематизації знаходження всіх можливих варіантів вирішення поставленої задачі на основі аналізу структури досліджуваної системи/об'єкта. Широко використовується в прогнозуванні складних процесів та створенні принципово нових сценаріїв

Фріц Цвікі створив цей метод та виділив його 5 основних кроків :

1. Формулювання та визначення проблеми;
2. Ідентифікація та характеристика всіх параметрів;
3. Побудова багатовимірної матриці (морфологічної таблиці), комбінації якої міститимуть усі можливі рішення;
4. Оцінка результатів на основі здійсненності та досягнення бажаних цілей;
5. Поглиблений аналіз кращих можливостей з урахуванням наявних ресурсів.

Мета ММА: організувати інформацію відповідним та корисним чином, щоб допомогти вирішити проблему або стимулювати нові способи мислення.

Основним об'єктом ММА є морфологічна таблиця. Вона складається з N характеристичних параметрів $F_i, i \in \overline{1,N}$. Кожному параметру F_i відповідає множина альтернатив $a_j^{(i)}, j \in \overline{1,n_1}$.

Зачасту параметрами є фактори, стан яких характеризує стан проблеми в цілому, а альтернативами є альтернативні стани відповідних факторів. Конфігурація морфологічної таблиці – сценарій.

Експертне оцінювання

Для деякої конкретної проблеми структура морфологічної таблиці визначається спеціалістами з передбачення. Для подальших розрахунків необхідно отримати початкові наближення $p_j^{`(i)}$ для ймовірностей альтернатив характеристичних параметрів, що відбувається за допомогою експертного оцінювання

Визначення оцінок експертами у конкретній області, базується на інтуїції отриманій шляхом досвіду у сфері. Фактично експерт має правильно оцінити вплив на той чи іншого фактор на об'єкт дослідження. Початкові наближення можна отримати використовуючи різні методи:

1. Однакові значення
2. Безпосереднє експертне оцінювання
3. Попарне порівняння

Експертне оцінювання, часто є єдиним джерелом інформації про об'єкт передбачення. Однак, експертні оцінки суб'єктивні і відображають реальний світ лише в деякому наближенні.

Перший етап морфологічного аналізу -

На першому етапі двоетапного ММА аналізуються неконтрольовані чинники. Для врахування зв'язків між параметрами морфологічної таблиці використовується числова матриця взаємної узгодженості. Кожній парі альтернатив $a_{j_1}^{i_1}$, $a_{j_2}^{i_2}$, а різних параметрів F_{i_1} , F_{i_2} присвоюється оцінка $c_{i_1 j_1 i_2 j_2} \in [-1, 1]$. У результаті цієї процедури формується матриця взаємозв'язків альтернатив.

		F_1				…	F_{N-1}			
		$a_1^{(1)}$	$a_2^{(1)}$	…	$a_{n_1}^{(1)}$	…	$a_1^{(N-1)}$	$a_2^{(N-1)}$	…	$a_{n_{N-1}}^{(N-1)}$
F_2	$a_1^{(2)}$	$c_{11,21}$	$c_{12,21}$	…	$c_{1n_1,21}$					
	$a_2^{(2)}$	$c_{11,22}$	$c_{12,22}$	…	$c_{1n_1,22}$					
	…	…	…	…	…					
	$a_{n_2}^{(2)}$	$c_{11,2n_2}$	$c_{12,2n_2}$	…	$c_{1n_1,2n_2}$					
…	…	…				…				
F_N	$a_1^{(N)}$	$c_{11,N1}$	$c_{12,N1}$	…	$c_{1n_1,N1}$	…	$c_{(N-1)1,N1}$	$c_{(N-1)2,N1}$	…	$c_{(N-1)n_{N-1},N1}$
	$a_2^{(N)}$	$c_{11,N2}$	$c_{12,N2}$	…	$c_{1n_1,N2}$		$c_{(N-1)1,N2}$	$c_{(N-1)2,N2}$	…	$c_{(N-1)n_{N-1},N2}$
	…	…	…	…	…		…	…	…	…
	$a_{n_N}^{(N)}$	c_{11,Nn_N}	c_{12,Nn_N}	…	c_{1n_1,Nn_N}		$c_{(N-1)1,Nn_N}$	$c_{(N-1)2,Nn_N}$	…	$c_{(N-1)n_{N-1},Nn_N}$

Експертні оцінки, отримані на попередньому етапі, не враховують взаємозв'язки між параметрами, визначені матрицею взаємозв'язків альтернатив параметрів, і тому є наближеними. Щоб отримати остаточні значення ймовірностей, необхідно розв'язати задачу розрахунку ймовірностей альтернатив параметрів.

Задача розрахунку ймовірностей альтернатив параметрів

Потрібно: розрахувати ймовірності $p_j^{(i)}$ настання кожної з альтернатив $a_j^{(i)}$. Для цього вирішується наступна система рівнянь Баєса

$$
\begin{cases}
p_1^{(1)} = \sum_{j_2=1}^{n_2}\sum_{j_3=1}^{n_3}\dots\sum_{j_N=1}^{n_N} P(\{a_1^{(1)}, a_{j_2}^{(2)}, a_{j_3}^{(3)}, \dots, a_{j_N}^{(N)}\} | a_{j_2}^{(2)}) p_{j_2}^{(2)}; \\
\dots \\
p_{n_1}^{(1)} = \sum_{j_2=1}^{n_2}\sum_{j_3=1}^{n_3}\dots\sum_{j_N=1}^{n_N} P(\{a_{n_1}^{(1)}, a_{j_2}^{(2)}, a_{j_3}^{(3)}, \dots, a_{j_N}^{(N)}\} | a_{j_2}^{(2)}) p_{j_2}^{(2)}; \\
p_1^{(2)} = \sum_{j_1=1}^{n_1}\sum_{j_3=1}^{n_3}\dots\sum_{j_N=1}^{n_N} P(\{a_{j_1}^{(1)}, a_1^{(2)}, a_{j_3}^{(3)}, \dots, a_{j_N}^{(N)}\} | a_{j_3}^{(3)}) p_{j_3}^{(3)}; \\
\dots \\
p_{n_2}^{(2)} = \sum_{j_1=1}^{n_1}\sum_{j_3=1}^{n_3}\dots\sum_{j_N=1}^{n_N} P(\{a_{j_1}^{(1)}, a_{n_2}^{(2)}, a_{j_3}^{(3)}, \dots, a_{j_N}^{(N)}\} | a_{j_3}^{(3)}) p_{j_3}^{(3)}; \\
\dots \\
p_1^{(N)} = \sum_{j_1=1}^{n_1}\sum_{j_2=1}^{n_2}\dots\sum_{j_{N-1}=1}^{n_{N-1}} P(\{a_{j_1}^{(1)}, a_{j_2}^{(2)}, a_{j_3}^{(3)}, \dots, a_1^{(N-1)}\} | a_{j_1}^{(1)}) p_{j_1}^{(1)}; \\
\dots \\
p_{n_N}^{(N)} = \sum_{j_1=1}^{n_1}\sum_{j_2=1}^{n_2}\dots\sum_{j_{N-1}=1}^{n_{N-1}} P(\{a_{j_1}^{(1)}, a_{j_2}^{(2)}, a_{j_3}^{(3)}, \dots, a_{n_N}^{(N-1)}\} | a_{j_1}^{(1)}) p_{j_1}^{(1)}; \\
\sum_{j=1}^{n_1} p_j^{(1)} = 1; \\
\dots \\
\sum_{j=1}^{n_N} p_j^{(N)} = 1.
\end{cases}
$$

Після цього, ми отримаємо морфологічну таблицю, що містить у собі оцінки альтернатив, які враховують взаємозв'язки параметрів системи. Ці значення використовуються в подальшому аналізі, виборі конфігурацій, які задовольняють певний критерій. В нашому випадку отримані значення слугуватимуть в якості вхідних даних для другого етапу двоетапної процедури морфологічного аналізу.

Другий етап морфологічного аналізу

Другий етап дослідження полягає в синтезі рішень, які найбільш ефективно враховувати в умовах сукупності можливих реалізацій об'єкта, визначених на першому етапі. Специфіка другого етапу МММА полягає в тому, що вибір альтернатив параметрів МТ стратегій залежить не від випадкових зовнішніх факторів, а від особи, що приймає рішення, тому немає сенсу говорити про ймовірність вибору альтернатив. Тому на другому етапі для оцінки альтернатив і конфігурацій використовується величина очікуваної результативності, тобто вірогідності того, що вибір цієї альтернативи або конфігурації призведе до бажаних результатів.

Таким чином, будуються дві пов'язані морфологічні таблиці: сценаріїв(перший етап), стратегій(другий етап). Параметри морфологічної таблиці стратегій залежать від морфологічної таблиці сценаріїв, побудованій на першому етапі. Для того, щоб врахувати цей зв'язок необхідно побудувати матрицю зв'язків альтернатив параметрів, де зв'язок між параметрами є одностороннім. Маємо що, кожній парі альтернатив $a_{j_1}^{i_1}$, $a_{j_2}^{i_2}$ різних параметрів F_{i_1} , F_{i_2} таблиць першого та другого етапів МММА присвоюється оцінка $c_{i_1 j_1 i_2 j_2} \in [-1, 1]$

Матриця заповнюється експертами за допомогою процедури, аналогічної до заповнення матриці взаємної узгодженості

Задача морфологічного дослідження другого етапу

Задачу, що виникає на другому етапі морфологічного дослідження, можна сформулювати наступним чином:

Дано:

- морфологічна таблиця, що складається з МТ першого та другого етапів. Таблиця містить множину характеристичних параметрів $F = \{F_i \mid i \in \overline{1, N+N'}\}$, кожний параметр F_i описується множиною альтернатив $A_i = \{a_j^{(i)} \mid j \in \overline{1, n_i}\}$;
- результати розрахунку ймовірностей альтернатив МТ сценаріїв $p_j^{(i)}, i \in \overline{1, N}, j \in \overline{1, n_i}$;
- значення зв'язків пар альтернатив параметрів МТ сценаріїв і стратегій $\{c_{i_1 j_1 i_2 j_2} \mid i_1 \in \overline{1, N}, i_2 \in \overline{N+1, N+N'};\ j_1 \in \overline{1, n_{i_1}};\ j_2 \in \overline{1, n_{i_2}}\}$.

Знайти:

- розрахувати оцінки результативності $R_j^{(i)}$ кожної з альтернатив параметрів МТ стратегій $a_j^{(i)}, i \in \overline{N+1, N+N'}, j \in \overline{1, n_i}$ в умовах ситуації, заданої МТ сценаріїв;
- розрахувати оцінки результативності $R\{s_l\}$ конфігурацій s_l, породжених МТ стратегій, в умовах ситуації, заданої МТ сценаріїв.

Врахуємо наступне:

Вибір тих чи інших альтернатив параметрів МТ стратегій є прийняттям рішення в умовах можливих станів навколишнього середовища, які можуть настати з певною ймовірністю. Для визначення очікуваної результативності необхідно розглянути всі можливі конфігурації МТ сценаріїв, враховуючи результативність розглядуваної альтернативи в умовах кожної з конфігурацій

Структура додатку

Інтерфейс з усіма необхідними елементами для взаємодії та здійснення обчислень

АПІ-сервіс, який взаємодіє і з інтерфейсом і з базою даних, та проводить ряд важливих перевірок

База даних, яка зберігає дані про користувача, уведені системи та усі обчислення

Основні інструменти розробки

Бібліотека React

Програмний каркас Express

Документоорієнто вана БД Mongoose

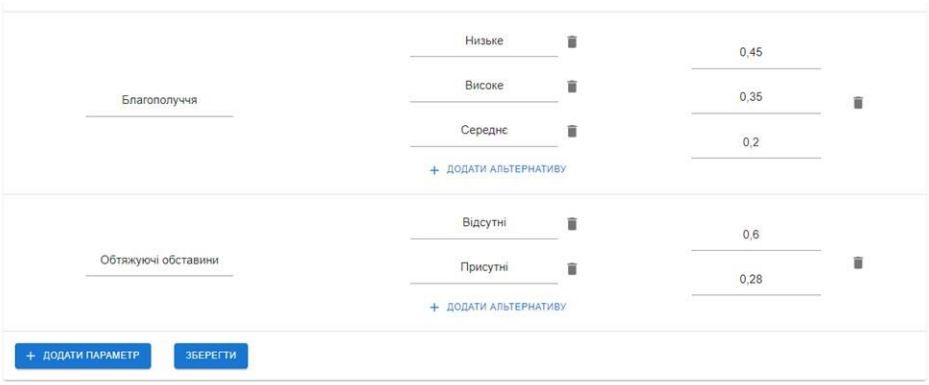
Приклад роботи додатку

Введення параметрів, альтернатив та значень задачі

Введіть дані

Parameter	Alternatives	Values
Вид власності	Державна	0,3
	Приватна	0,7
	+ ДОДАТИ АЛЬТЕРНАТИВУ	
Період вчинення	Робочий	0,2
	Позаробочий	0,8
	+ ДОДАТИ АЛЬТЕРНАТИВУ	
Охоронні системи	Датчики руху	0,28
	Камери спостереження	0,4
	Відсутня	0,32
	+ ДОДАТИ АЛЬТЕРНАТИВУ	

Введення параметрів, альтернатив та значень задачі



Матриця взаємозв’язків параметрів

Матриця взаємозв'язків

	Державна	Приватна	Робочий	Позаробочий	Датчики руху	Камери спостереження	Відсутня	Низьке	Високе	Середнє	Відсутні	Присутні
Державна	0	0	0,25	0,14	-0,13	0,2	-0,17	0,14	0,2	0,14	-0,15	0,16
Приватна	0	0	0,25	-0,2	-0,35	-0,19	-0,16	0,25	0,14	0,28	-0,12	-0,08
Робочий	0,25	0,25	0	0	0,36	-0,19	0,16	0,22	0,09	0,07	0,22	0,06
Позаробочий	0,14	-0,2	0	0	0,23	-0,17	-0,12	0,11	-0,1	0,13	0,1	0,01
Датчики руху	-0,13	-0,35	0,36	0,23	0	0	0	-0,22	0,08	-0,15	-0,12	0,13
Камери спостереження	0,2	-0,19	-0,19	-0,17	0	0	0	0,19	-0,24	-0,12	-0,19	-0,19
Відсутня	-0,17	-0,16	0,16	-0,12	0	0	0	0,01	0,03	0,08	-0,11	0,07
Низьке	0,14	0,25	0,22	0,11	-0,22	0,19	0,01	0	0	0	0,1	-0,12
Високе	0,2	0,14	0,09	-0,1	0,08	-0,24	0,03	0	0	0	0,11	-0,1
Середнє	0,14	0,28	0,07	0,13	-0,15	-0,12	0,08	0	0	0	-0,1	0
Відсутні	-0,15	-0,12	0,22	0,1	-0,12	-0,19	-0,11	0,1	0,11	-0,1	0	0
Присутні	0,16	-0,08	0,06	0,01	0,13	-0,19	0,07	-0,12	-0,1	0	0	0

ОЦІНКИ АЛЬТЕРНАТИВ

Результати першого етапу

Ймовірності альтернатив разом з конфігураціями

Вид власності	Період вчинення	Охоронні системи	Благополуччя	Обтяжуючі обставини	C	p	pC	P
Державна	Робочий	Датчики руху	Низьке	Відсутні	1.0038160000000004158	0.0045360000000000097	0.00455330937600000128	0.0057769981092(
Державна	Робочий	Датчики руху	Низьке	Присутні	1.2227142399999973020	0.0021168000000000071	0.0025882415032320015	0.0032849607874(
Державна	Робочий	Датчики руху	Високе	Відсутні	1.0129415999999999755	0.0035280000000000032	0.0035736579648000036	0.0045356379099!
Державна	Робочий	Датчики руху	Високе	Присутні	1.2505031999999981491	0.0016464000000000034	0.0020588284684799998	0.0026130369900:
Державна	Робочий	Датчики руху	Середнє	Відсутні	0.82130399999999992300	0.0020160000000000043	0.0016557488640000031	0.0021014538578!
Державна	Робочий	Датчики руху	Середнє	Присутні	1.3894479999999979434	0.0009408000000000021	0.0013071926784000001	0.0016590711047(

ОБЧИСЛИТИ ОЦІНКИ

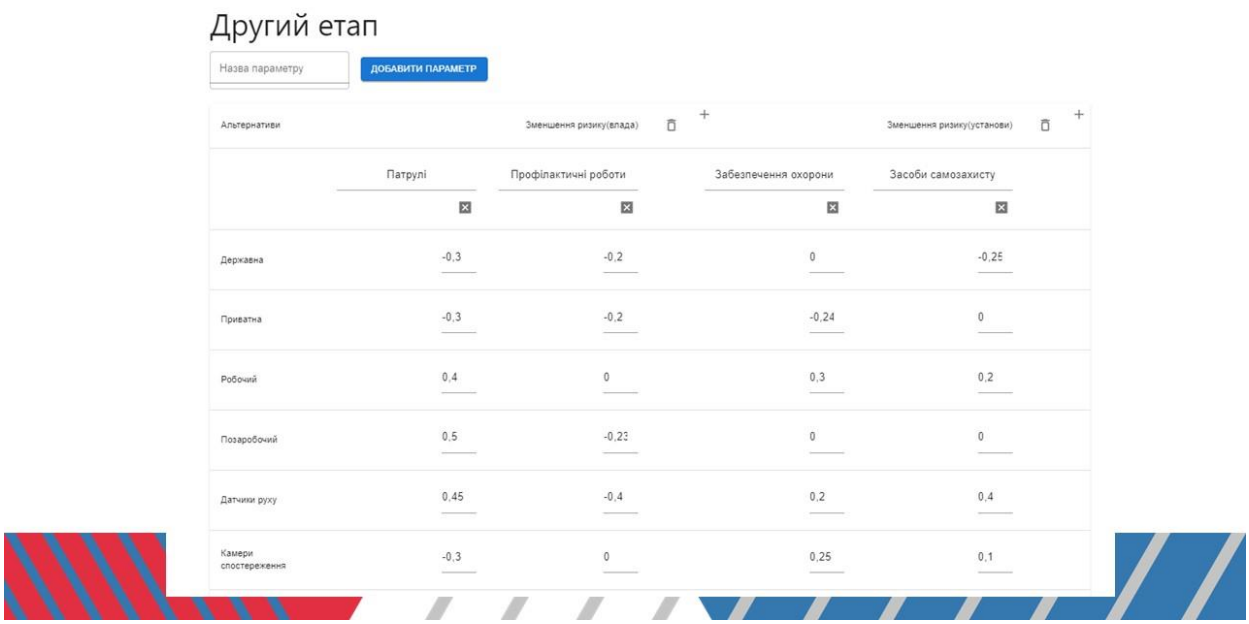
Результати першого етапу

Оцінки альтернатив

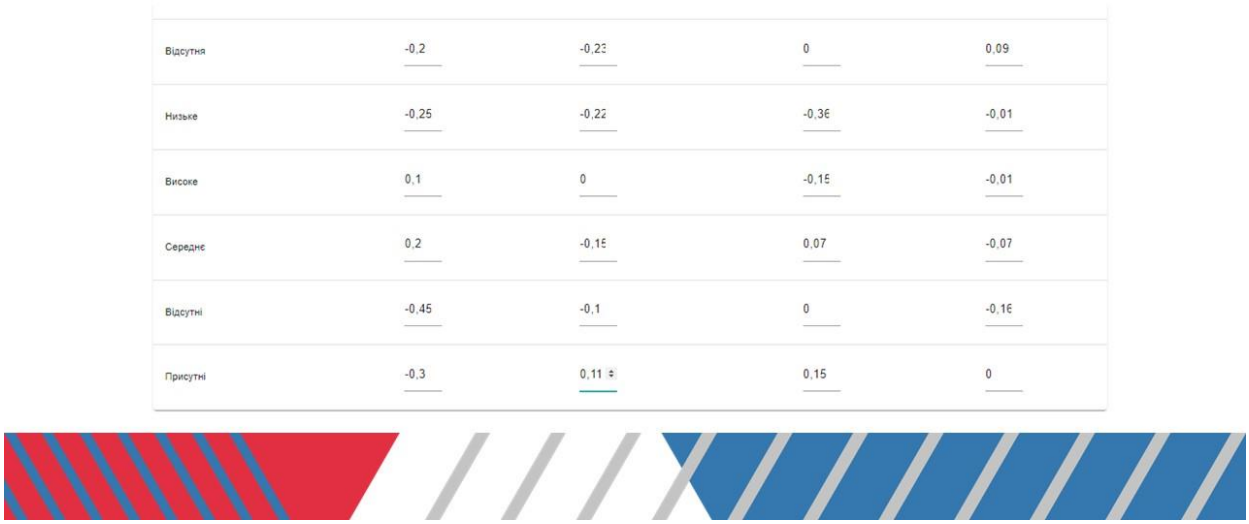
Parameter	Alternative	Probability
Вид власності	Державна	0.31145234245703274
	Приватна	0.6885476575429673
Період вчинення	Робочий	0.21410751279224127
	Позаробочий	0.785892487207759
Охоронні системи	Датчики руху	0.2990070676154903
	Камери спостереження	0.36302288708348895
	Відсутня	0.3379700453010209
Благополуччя	Низьке	0.4555422492010418
	Високе	0.3590159031220145
	Середнє	0.18544184767694386
Обтяжуючі обставини	Відсутні	0.6780571436979568
	Присутні	0.3219428563020433

ДРУГИЙ ЕТАП

Введення параметрів, альтернатив та значень другого етапу



Введення параметрів, альтернатив та значень другого етапу



Результати другого етапу

Очікуваний результат

Зменшення ризику(влада)

Альтернатива	Ймовірність
Патрулі	0.6179836007043031
Профілактичні роботи	0.3816578129972149

Результати другого етапу

Очікуваний результат

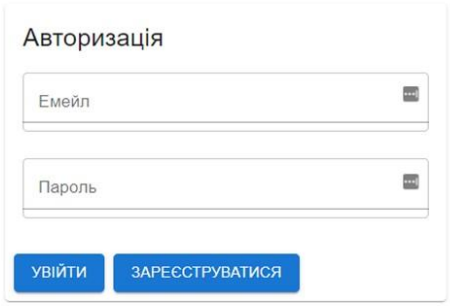
Зменшення ризику(установи)

Альтернатива	Ймовірність
Забезпечення охорони	0.416777745887499
Засоби самозахисту	0.41537274061732826

Висновки

В ході виконання роботи було розглянуто модифікований метод морфологічного аналізу та його застосування. Опісля було створено архітектуру програмного забезпечення, яке б відтворювало роботу методу. Далі почався етап реалізації модулів додатку, та перевірка на коректне функціонування. Кінцевим етапом стало налаштування спільної роботи елементів та стилізація продукту

Безумовно, інтерфейс взаємодії користувача для здійснення морфологічного аналізу прикладний та корисний застосунок. Більше того, це лише частину яку бачить користувач, хоча позаду відбувається набагато складніші процеси які і дозволяють усе це відобразити на екрані



Подальший розвиток

- Забезпечення порівняння систем
- Збільшення швидкості роботи додатку
- Забезпечення інтуїтивної зрозумілості використання

Посилання на інструменти, які дозволили реалізувати проект

React

Express

MongoDB

MUI

Дякую за увагу!