**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Сумський державний університет**
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Оксана ШОВКОПЛЯС
_____
(підпис)
грудня 2024 р.

# КВАЛІФІКАЦІЙНА РОБОТА

**на здобуття освітнього ступеня магістр**

зі спеціальності 122 «Комп'ютерні науки»
освітньо-професійної програми «Інформатика»
на тему: Інтелектуальна технологія планування розгортання допомоги
здобувача групи ІН.м-32 Ільченка Єгора Валентиновича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Єгор ІЛЬЧЕНКО
(підпис)

Керівник
доцент,
к.т.н., _____ Борис КУЗІКОВ
(підпис)

**Суми – 2024**

**Сумський державний університет**
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Оксана ШОВКОПЛЯС
(підпис)

# ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

**на здобуття освітнього ступеня магістра**

зі спеціальності 122 «Комп'ютерні науки», освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-32 Ільченка Єгора Валентиновича

1. Тема роботи: Інтелектуальна технологія планування розгортання допомоги
затверджую наказом по СумДУ від *«03» грудня 2024 року №1257-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 06 грудня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
*1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються при розробці систем розподілення вантажів 3) Розробка інтелектуальної системи планування розгортання допомоги 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із зазначенням розділів проекту, що стосується їх

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

# КАЛЕНДАРНИЙ ПЛАН

| № п/п | Назва етапів кваліфікаційної роботи | Термін виконання | Примітка |
|---|---|---|---|
| *1* | *Аналіз проблеми предметної області, постановка й формування завдань дослідження* | | |
| *2* | *Огляд технологій, що використовуються при розробці систем розподілення вантажів* | | |
| *3* | *Розробка інтелектуальної системи планування розгортання допомоги* | | |
| *4* | *Аналіз отриманих результатів* | | |
| *5* | *Оформлення пояснювальної записки до кваліфікаційної роботи* | | |

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

# АНОТАЦІЯ

**Записка**: 70 стр., 6 рис., 3 табл., 29 формул, 1 додаток, 22 використаних джерел.

**Обґрунтування актуальності теми роботи** – Зростання попиту на гуманітарну допомогу в постраждалих зонах, ускладнює гуманітарні логістичні операції. Це вимагає ефективного управління обмеженими ресурсами для забезпечення невідкладних потреб постраждалих.

**Об'єкт дослідження** — Процеси управління розподілом та постачанням запасів.

**Предмет дослідження**. Стохастична динамічна задача інвентарного розподілу, що зосереджена на управлінні постачанням гуманітарної допомоги в умовах невизначеності попиту та пропозиції.

**Мета роботи** — розробка технології розподілення запасів з урахуванням динамічного середовища з використанням підходів наближеного динамічного програмування (ADP).

**Методи дослідження** — алгоритми розв'язання задач наближеного динамічного програмування, системи керування базами даних, інструменти, методи побудови лінійної регресії, методи вирішення задачі змішаних цілих (MIP) і прогнозування подій та інструменти побудови математичних моделей.

**Результати** — розроблено інформаційну технологію, яка надає змогу проводити моніторинг гуманітарних потреб та на основі цих даних проводить оптимізацію розподілення запасів. Проведено тестування розробки на тестових даних м. Суми, для координат у місті згенерованих за нормальним розподілом.

**Об'єкт дослідження**.

**Новизна**. Використано розв'язання на основі апроксимованого динамічного програмування з задіянням підходу декомпозованої лінійної апроксимаційної функції значення (DL-VFA) із використанням даних з попередньо проведеного моніторингу потреб і визначених оптимальних точок дистриб'юції. Функція дозволяє більш ефективно враховувати невизначеності процесу динамічного розподілу.

**Структура роботи** складається зі вступу, аналітичного огляду, постановки задачі, вибору методу розв'язання поставленої задачі, реалізації алгоритму оптимізації, висновків, списку використаних джерел та додатків.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, APPROXIMATE DYNAMIC PROGRAMMING, SKLEARN, KMEANS , JAVA,PYTHON, NUMPY, MIP, SPRING, ЛІНІЙНА РЕГРЕСІЯ

# ЗМІСТ

# ВСТУП

Сьогодні інформаційні технології є важливим інструментом, який застосовується у різних сферах життя для автоматизації рутинних процесів, скорочення часу на їх виконання та забезпечення зручного доступу до необхідних послуг.[1] В умовах надзвичайних ситуацій, таких як катастрофи, такі технології набувають особливого значення, оскільки сприяють швидшій взаємодії, збору та обробці даних про потреби населення. Створення інформаційної системи для моніторингу гуманітарних потреб є актуальним завданням, яке може вплинути ефективність надання допомоги, дозволяючи відстежувати потреби, оптимально розподіляти запаси та зменшувати відстань до пунктів видачі гуманітарної допомоги для постраждалих.

Ефективне управління гуманітарною допомогою є критичним аспектом реагування на катастрофи, які супроводжуються різкими коливаннями в потребах постраждалих регіонів та обмеженими ресурсами. В умовах катастроф виникає необхідність оптимізації процесу розподілу обмежених запасів між кількома районами, що постраждали, з урахуванням таких факторів, як невизначеність попиту та логістичні обмеження. Традиційні підходи до розподілу ресурсів часто виявляються неефективними у таких ситуаціях через статичність і недостатню гнучкість. Тому актуальним завданням є розробка динамічних моделей розподілу, які дозволяють швидко адаптуватися до змін у потребах та наявності ресурсів. Розробка направлена на оптимізацію стохастичних моделей, здатних забезпечити справедливий і своєчасний розподіл допомоги, мінімізуючи наслідки дефіциту для постраждалих громад.

# 1. АНАЛІТИЧНИЙ ОГЛЯД

Проаналізуємо інструменти, націлені на надання гуманітарної допомоги і методи оптимізації її розподілення в них.

## 1.1. Підтвердження доцільності розробки системи

Основними завданнями гуманітарної допомоги є порятунок людського життя, забезпечення базової гігієни та медичної допомоги, а також задоволення основних потреб людини [2]. Додаток покликаний спростити механізм надання останніх двох видів допомоги.

Згідно з доповіддю Координаційного штабу від 16.04.2022 р. [3], продуктові набори було доставлено до дев'яти областей і міста Київ, загалом 8 321 605 наборів, що свідчить про значний масштаб допомоги з боку держави і міжнародних організацій. Розташування пунктів видачі в зручних для громадян місцях і оптимізація розподілу ресурсів у них сприяє ефективнішому забезпеченню потребуючих.

У звіті «Механізми надання державою гуманітарної допомоги в умовах воєнного стану» за червень 2022 р. виділено ключові проблеми, зокрема:

- недостатню координацію між місцевою владою та волонтерськими ініціативами;
- змішування процесів постачання на різних рівнях, що спричиняє відсутність єдиного бачення роботи системи;
- відсутність системи визначення та аналізу потреб, що особливо ускладнює ситуацію в зонах бойових дій;
- непрозорий механізм розподілу допомоги на складах та серед отримувачів [3].

За набуття чинності порядку №953 від 2023р Алгоритм отримання гуманітарної допомоги наступний: письмова пропозиція донора, надання

письмової згоди, підписання первинних документів, реєстрація отримувачів, відображення у бухобліку (облік гуманітарної допомоги), передача набувачам, звітування. [4]

До того ж на рівні закону запроваджується використання автоматизованої системи реєстрації гуманітарної допомоги. Будь-який гуманітарний вантаж тепер може бути оформлено виключно на організацію/установу, яка включена до Єдиного реєстру отримувачів гуманітарної допомоги.

Цей закон частково вирішує проблему зі звітністю, проте, все це свідчить про відсутність ініціативи створення єдиної спрощеної системи інформування вразливих верств населення про масштабні надходження з боку державних або міжнародних установ і спроби їх розподілу в зручному порядку. Основний напрямок насьогодні – спрямування на індивідуальних осіб і максимальна бюрократизація запитів на гуманітарну допомогу. Розроблювальна система ж має на меті гнучкість і спрощену систему реєстрації, бо основне її завдання – моніторинг потреб і, що важливо, оптимізація доставки і розподілу ресурсів згідно з результатами моніторингу.

Окремо варто приділити увагу важливості динамічного розподілу ресурсів. За даними Глобального гуманітарного огляду Організації Об’єднаних Націй, у 2023 році гуманітарної допомоги та захисту потребували рекордні 339 мільйонів людей, що є значним зростанням у порівнянні зі 136 мільйонами у 2018 році. Зважаючи на зростання частоти та масштабів катастроф, гуманітарна логістика є критично важливою для забезпечення населення необхідними засобами для виживання та полегшення страждань, викликаних дефіцитом товарів та послуг. Однак, через раптовий ріст попиту на етапі реагування на катастрофу, що триває від кількох днів до кількох тижнів, можливості агентств часто обмежені, що спричиняє дефіцит запасів. Отже, для ефективного розподілення ресурсів важливо враховувати часову і просторову динаміку та забезпечити баланс між благополуччям постраждалих, витратами на транспортування та обмеженими фінансовими ресурсами.

Оскільки невизначеність попиту та умов у зоні катастроф не дозволяє використовувати фіксовані плани розподілу, необхідно застосовувати адаптивні підходи.

Таким чином, виникає необхідність у розробці стохастичних моделей динамічного розподілення, які б враховували як часово-просторові характеристики, так і витрати на недоотримання допомоги. Це дозволить значно оптимізувати гуманітарну логістику, забезпечуючи більш ефективний розподіл ресурсів серед постраждалих і мінімізуючи їхні втрати. [5]

Отже, розробка інформаційної системи є актуальною і доцільною для вирішення виявлених проблем.

## 1.2. Аналіз існуючих рішень

Для оцінки корисності функціоналу пропонованої системи варто проаналізувати існуючі рішення у сфері надання гуманітарної допомоги. Розглянуті веб-додатки спрямовані на полегшення процесів збору та доставки допомоги, хоча переважна більшість з них орієнтована на забезпечення фінансової підтримки. Так, платформи United24 та #HelpUkraineWithCrypto спрощують переказ та контроль за використанням коштів, призначених для підтримки військових і цивільних потреб. Help.gov.ua забезпечує збір коштів для допомоги з-за кордону, розділяючи потреби на загальні категорії, а #Є_Допомога координує фінансову підтримку і допомогу для переміщення. Help Ukraine Center та СпівДія організовують постачання гуманітарної та медичної допомоги, об'єднуючи волонтерські і державні ініціативи, а Prozorro+ допомагає у пошуку постачальників товарів та фінансування від міжнародних донорів.[3]

Водночас деякі платформи не відповідають завданням моніторингу гуманітарних потреб, оскільки спрямовані здебільшого на підтримку матеріальних та логістичних ресурсів (наприклад, United24, #HelpUkraineWithCrypto, Help Ukraine Center, Залізна_допомога, Prozorro+).

Система «Залізна допомога», наприклад, спеціалізується на транспортуванні гуманітарних вантажів залізничними шляхами, але не передбачає прямої взаємодії з цивільним населенням. Таким чином, у проаналізованих платформах відсутній фокус на моніторинг потреб у гуманітарній допомозі серед постраждалих громад, а також на оптимізацію розподілу обмежених ресурсів у реальному часі. Це свідчить про необхідність розробки нової системи, яка не лише збирає інформацію про потреби, але й дозволяє оперативно і ефективно розподіляти наявні запаси серед постраждалих регіонів.

Варто окремо згадати про державну ініціативу створення від 2023р «Автоматизованої системи реєстрації гуманітарної допомоги», яка націлена на забезпечення прозорого обліку отримувачів гуманітарної допомоги та вантажів. В ній є плюс – централізований контроль. Але все ж таки, не підходить під наші задачі, оскільки має іншу мету – бюрократизація процесів отримання гуманітарної допомоги задля прозорості та звітності її розподілення, що є, безумовно, позитивним моментом.

Детальне порівняння наведених систем наведено у таблиці 1.1.

Таблиця 1.1 – аналіз особливостей наявних систем

| Особливості системи / Наявні Системи | Категорізація допомоги (ліки, їжа, одяг) | Є можливість надання допомоги | Націлена на збір індивідуальних потреб | Має систему моніторингу | Має систему оптимізації динамічного розподілення ресурсів | Націлена на централізованість і підзвітність |
|---|---|---|---|---|---|---|
| **#ЛогістикаРазом** | + | + | + | - | ? | + |
| **Паляниця.Інфо** | + | + | + | - | - | - |
| **Автоматизована система реєстрації гуманітарної допомоги** | + | + | + | - | ? | + |
| **СпівДія** | + | + | + | - | - | - |
| **Help.gov.ua** | + | - | + | - | - | + |
| **#Є_Допомога** | + | + | + | - | - | + |
| **Проєкт “Продуктовий набір”** | - | + | + | - | ? | - |
| **ІС «Convenient aid»** | - | - | - | + | + | + |

Аналіз існуючих чат-ботів і систем для підтримки волонтерської роботи та надання допомоги показав, що вони не відповідають нашим потребам з точки зору моніторингу гуманітарної ситуації. Наприклад, *Odeshyna Help Bot* обмежений регіонально, а *@saveua_bot*, *@share_help_bot*, *@nampodorozi_bot*, *@turbotnyk_bot* та *@DopomogaRazomBot* лише сприяють контактам для отримання допомоги, проте не можуть здійснювати процеси збору потреб і моніторингу.

Тобто існуючі системи для пошуку допомоги, координації волонтерів та збору потреб здебільшого спрямовані на індивідуальну підтримку і не мають функцій моніторингу та оптимізації розподілу допомоги. Їм бракує інструментів для ефективного визначення потреб на рівні міста чи оптимального розташування наборів. Наша система покликана вирішити цю проблему, забезпечуючи централізований моніторинг і оптимізацію розподілу гуманітарної допомоги за актуальними даними.

### 1.3. Аналіз існуючих рішень для оптимізації розподілу гуманітарної допомоги

У роботі «Artificial Intelligence as Decision Aid in Humanitarian Response» [6] підсумовується, що інтеграція ШІ, дистанційного зондування (RS) та великих даних значно покращує процеси прийняття рішень у гуманітарних операціях, підвищує ефективність завдяки автоматизації, надає цінні теоретичні інсайти, а також демонструє практичну користь у реальних ситуаціях, створюючи основу для подальших досліджень та розвитку цих технологій у кризовому управлінні. Це свідчить про те, що при розробці «Інтелектуальної технології планування розгортання допомоги» варто розглянути існуючі рішення і підходи оптимізації розгортання допомоги з використанням технологій машинного навчання.

Розглянемо статтю «A Decision-Making Model to Determine Dynamic Facility Locations for a Disaster Logistic Planning Problem Using Deep Learning» [7], де для створення оптимізованої моделі управління розподілом допомоги при катастрофах, яка включає розміщення центрів розподілу, прогнозування потреб, моделювання заборонених маршрутів і ефективний розподіл гуманітарних вантажів розглядаються наступні підходи:

- Метод K-Means - алгоритм кластеризації використовується для прогнозування місць розташування центрів розподілу після катастроф.

- Штучні нейронні мережі для прогнозування запитів на допомогу навколо створених центрів розподілу.
- Модель заборонених маршрутів, яка призначена для визначення дозволених і заборонених маршрутів для розподілу гуманітарної допомоги.
- Змішане нелінійне багатокритеріальне програмування: Цей підхід застосовується для розв'язання задачі вибору маршрутів і місць розташування (DLRP). Такий підхід дозволяє оптимізувати логістику надання допомоги при катастрофах у комплексному вигляді.

Ефективність системи може варіюватися в залежності від місцевих умов та особливостей логістичної інфраструктури, проте на деякі підходи є універсальними та можуть бути корисними при розробці технології планування розгортання допомоги.

В статті «The Stochastic Dynamic Post-Disaster Inventory Allocation Problem with Trucks and UAVs» [5] було представлено два методи навчання з підкріпленням, один для найвищої ефективності, інший для кращої масштабованості. В ній пропонується використання декомпозованої лінійної апроксимації функції вартості (DL-VFA), яка розкладає задачу на просторово-часові функції вартості, забезпечуючи високу ефективність і відмінну масштабованість, при цьому комбінований розподіл постачань у кожному періоді вирішується шляхом розв'язання змішаної цілочислової програми (MIP), яка включає просторово декомпозовані функції вартості відповідного періоду. Або ж апроксимація функції вартості за допомогою нейронних мереж (NN-VFA), яка захоплює просторово-часові та нелінійні ефекти в одній нейронній мережі, що інтегрується в MIP, використану для вирішення одноетапної комбінаторної задачі розподілу. В роботі буде використано саме ці алгоритми з огляду на доцільність їх використання для нашої системи, та високої швидкодії.

## 1.4. Постановка задачі

Центральний склад (CW) є основним пунктом прийому допомоги в зоні, в якій фіксується її необхідність та розподіляє ресурси по локальних центрах розподілу в окремих районах, визначених за допомогою моніторингу. З локальних центрів розподілу виконується останній етап доставки допомоги безпосередньо бенефіціарам у районах. Мета роботи полягає в оптимізації процесу розподілу ресурсів з центрального складу по пунктів видачі, щоб забезпечити максимально ефективне й швидке задоволення потреб постраждалих.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1) Визначити основні технології, які будуть задіяні при розробці
2) Розробити чітку архітектуру додатку
3) Розробити основні модулі моніторингу
4) Визначити і впровадити алгоритм динамічної оптимізації розподілення гуманітарної допомоги орієнтуючись на навчання з підкріпленням.
5) Розробити зручне відображення усіх компонентів системи

# 2. ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

## 2.1. Вибір архітектури

Монолітна архітектура характеризується тісною інтеграцією компонентів, що забезпечує простоту розгортання та відладки. Однак її недоліки включають труднощі з масштабуванням, оновленням технологій та гнучкістю, оскільки будь-яка зміна потребує редагування всієї системи.

Мікросервісна архітектура, на відміну від монолітної, передбачає розподіл на окремі сервіси, що спілкуються між собою через повідомлення. Цей підхід забезпечує більшу гнучкість, масштабованість та незалежність технологій для кожного сервісу, проте він ускладнює налаштування взаємодії між підсистемами та розгортання.

Серверлесс-архітектура зосереджена на розробці замість розгортання, пропонуючи переваги у вигляді гнучкості та абстракції від ОС, але має складнощі з масштабуванням та відладкою через обмеження хмарних сервісів.[8]

Зважаючи на переваги та обмеження кожної архітектури, для додатку можна обрати монолітну архітектуру, що спрощує розробку. Водночас, у майбутньому, з огляду на можливість масштабування, система може бути розділена на мікросервіси для більшої гнучкості та ефективності.

## 2.2. Вибір мов програмування та супутніх фреймворків

Посилаючись на рейтинг мов програмування від tiobe маємо наступні результати (рисунок 2.1):

| Nov 2024 | Nov 2023 | Change | Programming Language | Ratings | Change |
|---|---|---|---|---|---|
| 1 | 1 | | Python | 22.85% | +8.69% |
| 2 | 3 | ↑ | C++ | 10.64% | +0.29% |
| 3 | 4 | ↑ | Java | 9.60% | +1.26% |
| **4** | **2** | ↓ | **C** | **9.01%** | **-2.76%** |
| 5 | 5 | | C# | 4.98% | -2.67% |
| 6 | 6 | | JavaScript | 3.71% | +0.50% |

Рисунок 2.1– рейтинг мов програмування[9]

Індекс TIOBE є показником популярності мов програмування, який оновлюється щомісяця. Рейтинг формується на основі кількості кваліфікованих інженерів, навчальних курсів та постачальників послуг у світі. Для розрахунку використовуються дані з Google, Amazon, Wikipedia, Bing та інших джерел.

Використовуючи ці дані, можна робити висновки про величину спільноти певної мови програмування. Зазвичай, чим популярніша мова програмування – тим частіше і легше можна знайти рішення для проблеми, яка виникла під час написання коду. Більше того, цей рейтинг може свідчити про те, що на певній мові програмування деякі рішення можна розробити швидше ніж на інших. Відштовхуючись від цього, пропонується розробити систему з залученням декількох мов програмування, серверної частини – на Java, для відображення в браузері (клієнтської частини) використовувати Javascript а для системи оптимізації, з огляду на велику кількість корисних бібліотек, використати python.

Проте замало використовувати лише мови програмування. Якщо б увесь функціонал треба було створювати кожного разу з «чистого листа» розробка займала б фінансово-невигідну кількість часу.

Фреймворки та бібліотеки дозволяють уникнути повторного написання базового функціоналу, проте різниця між ними полягає в принципі інверсії контролю. При використанні бібліотеки розробник сам керує потоком виконання програми, звертаючись до модулів за необхідності. Натомість

фреймворк управляє потоком, надаючи місця для підключення коду, виконання якого відбувається у певних ситуаціях [10].

Одним із провідних фреймворків Java є Spring, який забезпечує комплексну модель програмування для корпоративних додатків. Spring підтримує інфраструктурний рівень застосунків, зосереджуючись на загальних аспектах додатків, що дозволяє командам фокусуватися на бізнес-логіці, мінімізуючи прив'язку до середовищ розгортання.

Spring складається з модулів, таких як Core, Beans, Context, Expression Language, Web MVC тощо, що групуються для різних завдань. У розробці системи доцільно використовувати модулі Spring Core, Beans, Context, Expression Language, Web MVC, а для конфігурації — Spring Boot, який автоматично налаштовує проєкт і полегшує підключення бібліотек. Для реалізації специфікації JPA буде використано ORM-бібліотеку Hibernate.

Для створення бота на Python буде застосовано фреймворк aiogram, а також бібліотеку geopy для визначення назв населених пунктів за координатами (реверсивне геокодування). Також для оптимізації процесу розподілення гуманітарної допомоги використаємо бібліотеки з готовими рішеннями для навчання з підкріпленням такі як pandas, numpy, tensorflow.

### 2.3. Вибір системи керування базою даних

Для основної бази даних використаємо саме реляційну базу даних, завдяки простоти запитів (SQL), широкій підтримці таких СКБД мовами програмування Java та Python, підтримці усього необхідного нам функціоналу, низькими обсягами займаної пам’яті та узгодженості. [11]

Для основної бази даних у розробці системи обрано реляційну базу даних PostgreSQL. Вона забезпечує весь необхідний функціонал для системи, зокрема підтримує ключові обмеження даних, паралельну обробку транзакцій, складні SQL-запити та різні типи даних. PostgreSQL є сумісною з мовами Java та Python, що полегшує інтеграцію.

Альтернативні СКБД, такі як Oracle та Microsoft SQL Server, надають розширені функції для великих комерційних систем, але вони не є оптимальними з огляду на простоту, ефективність і відсутність надлишкових функцій, які потрібні для нашої системи. SQLite, попри простоту, має обмежену функціональність, що не повністю відповідає вимогам. [12]

Також використано MongoDB (NoSQL) для зберігання сесій у Telegram-боті, що дозволяє гнучке управління неструктурованими даними, необхідними для підтримки сесійної логіки бота.

## 2.4. Проектування системи

Для проектування системи скористаємося наступними діаграмами:

- Use case
- Entity-Relationship

Побудуємо ER діаграму (рисунок 2.2)

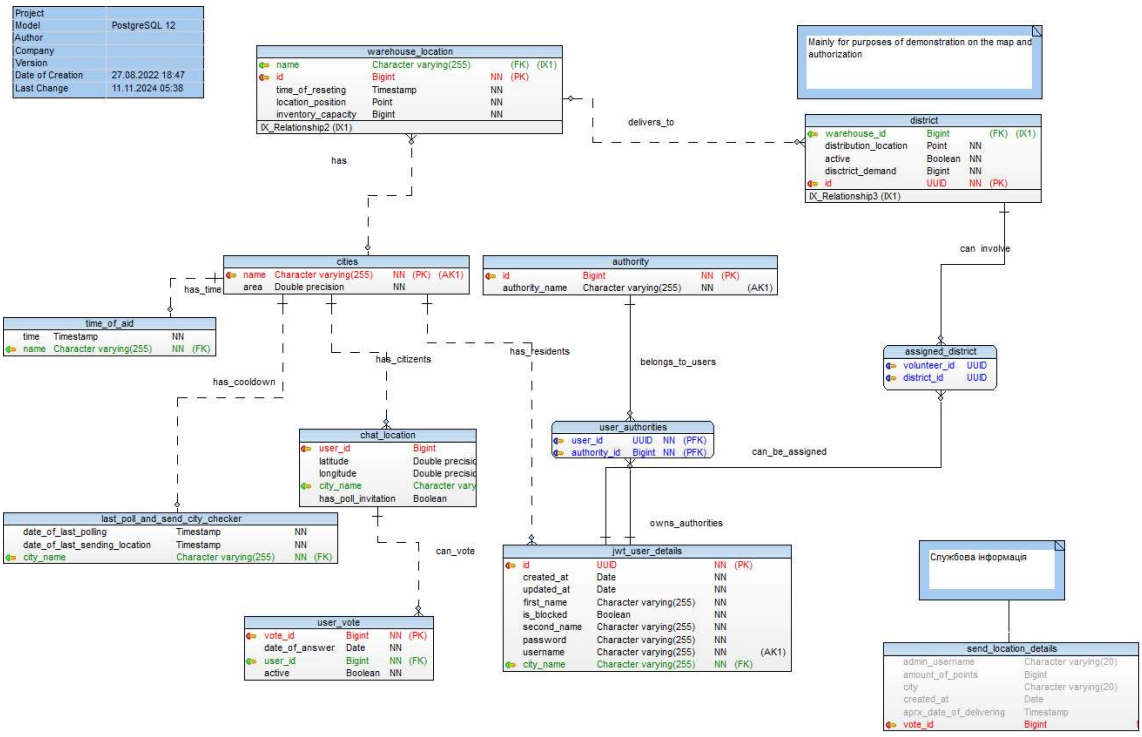


Рисунок 2.2 - ERD діаграма бази даних

На ній можна побачити, що Сутність *user_authorities* відповідає за автентифікацію та призначення ролей у веб-додатку, тоді як *chat_location* містить інформацію про користувачів, яких можна опитувати через Telegram API. *Send_location_details* служить для зберігання службової інформації. Додалися також і сутності warehouse_locations – для відображення центру зберігання ресурсів у певному місті і самі точки розподілу, або ж district з супутньою інформацією для відображення в клієнтській частині та впровадженні автентифікації і доступу до певних точок видачі.

## 2.5. Алгоритм підбору найкращих точок

Для впровадження алгоритму необхідно проаналізувати і вибрати спосіб знаходження відстані між двома точками, вираженими у широті й довготі, для цього є декілька способів:

Відстань між точками за формулою Гаверсина (Рівн. 2.1).

$$a = \sin^2(\Delta\varphi/2) + \cos\varphi1 \cdot \cos\varphi2 \cdot \sin^2(\Delta\lambda/2)$$

$$c = 2 \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)}) \qquad (2.1)$$

$$d = R \cdot c$$

, де $\varphi$ - latitude, $\lambda$ - longitude, R - радіус землі (radius = 6,371km);Кути необхідно вимірювати в радіанах.

За сферичним законом косинусів (рівн 2.2):

$$d = \operatorname{acos}(\sin\varphi1 \cdot \sin\varphi2 + \cos\varphi1 \cdot \cos\varphi2 \cdot \cos\Delta\lambda) \cdot R \qquad (2.2)$$

Для спрощення обчислень можна використовувати звичайну теорему Піфагора [14] (рівн. 2.3)

$$x = \Delta\lambda \cdot \cos \varphi m$$

$$y = \Delta\varphi \qquad (2.3)$$

$$d = R \cdot \sqrt{x^2 + y^2} \quad (2.2)$$

Розташування точок є попереднім, оскільки остаточний вибір залежить від локальних умов, таких як наявність під'їзних шляхів чи вимоги до безпеки, які складно заздалегідь врахувати. З огляду на це, застосування алгоритму кластеризації k-means є виправданим, оскільки це один із найпопулярніших та досліджених методів кластеризації. Алгоритм k-means, також відомий як узагальнений алгоритм Ллойда (GLA), належить до жорстких методів кластеризації, де як представники кластерів обирають конкретні точки (у нашому випадку — координати), а для обчислення відстаней між точкою X і її кластерним центром C використовують евклідові відстані або, за потреби, формулу Гаверсина (формула 2.1). [14]. Більше того, для кращої ефективності, буде використано модифікацію k-means++ — варіант алгоритму k-means, що використовує просту рандомізовану методику вибору початкових точок (seeding), завдяки чому досягається оптимальна кластеризація з обчислювальною складністю Θ(log(k)). Експерименти показують, що ця модифікація значно підвищує як швидкість, так і точність k-means[15].

Якщо потрібна вища точність і географічні координати слід розглядати як неевклідові дані, кластеризація стає складнішою і вимагає застосування додаткових алгоритмів. Серед найбільш популярних для таких задач, окрім ієрархічної кластеризації, є широко використовуваний алгоритм Partitioning

Around Medoids (PAM), який також просто називають k-medoids [16] та CLARA [17].

### 2.6. Серверна частина

Основний обмін даними між компонентами, що забезпечують моніторинг і опитування, здійснюється через REST — легкий у реалізації архітектурний стиль, який надає численні інструменти та підтримку документації для Java. REST є оптимальним для роботи з односторінковими додатками, такими як ReactJS-клієнт, адже він підтримує формат JSON, створений для JavaScript. [18]

Основні етапи створення REST-сервісу у Spring Boot включають:

- створення моделі (класів, що репрезентують дані);
- визначення відношень між ними (через JPA-анотації);
- створення інтерфейсів для роботи з базою даних (Repository);
- конфігурацію необхідних бінів;
- написання сервіс-класів для бізнес-логіки;
- розробку REST-контролера (згідно з MVC);
- обробку користувацьких помилок;
- захист (Spring Security) та тестування (TDD). [19]

Для захисту системи стандартна сесійна авторизація є недостатньою через використання ReactJS на клієнтській частині. Замість цього пропонується JWT (JSON Web Token) [20], що забезпечує компактне і захищене передавання даних про користувача. JWT містить закодовану інформацію про користувача (claims) і підпис, який можна верифікувати лише із знанням секретного ключа. Це дозволяє безпечно ідентифікувати користувача, забезпечуючи цілісність даних і захист від підробки.

## 2.7. Алгоритм оптимізації планування розгортання гуманітарної допомоги

Для досягнення задач оптимізації розподілення гуманітарної допомоги, яка базується на моделі ланцюга поставок гуманітарної допомоги (Balcik, Beamon, Smilowitz, 2008) [21], де центральний склад (CW) є основним пунктом отримання допомоги для постраждалої зони і розподіляє її до місцевих центрів у різних районах. Після надходження запасів у CW агентство приймає рішення про їх розподіл у дискретні часові моменти $t \in T = \{1,2,...,T\}$. Запаси можуть бути розподілені між районами $n \in N = \{1,2,...,N\}$ через різні транспортні засоби $k \in K = \{1,2,...,K\}$ (наприклад, вантажівки або БПЛА) або зберігатися для майбутніх періодів. Кожен транспорт має обмежену місткість $q_k$ і фіксовану вартість $c_{nk}$, незалежну від обсягу вантажу. Розподілені запаси $x_{tnk}$ поповнюють складські запаси району $I_{tn}$, які споживаються відповідно до випадкового попиту $\hat{d}_{tn}$. У момент часу t інформація про потреби та рівень запасів в районах оновлюється за допомогою звітів польових працівників, що дозволяє точніше прогнозувати попит $d_{t,t+1,n}$ на наступний період. Побачити це можна на наступному рисунку (рис 2.3)

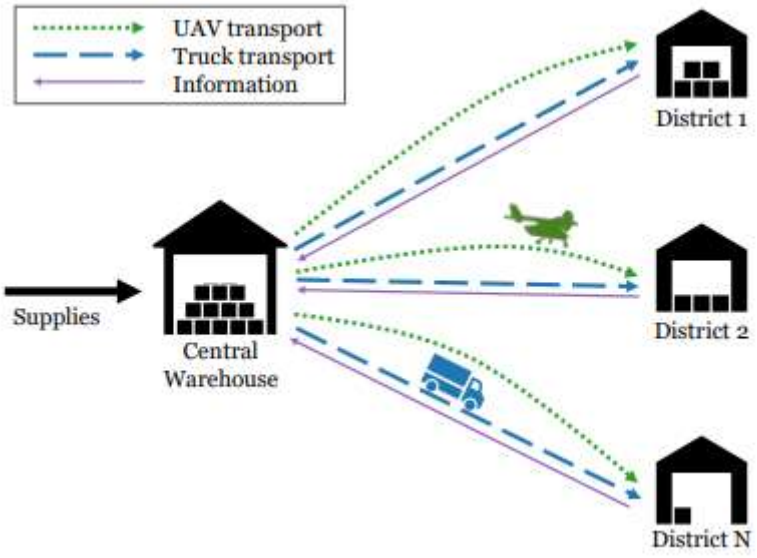


Рисунок 2.3 – розподіл гуманітарної допомоги[5]

Модель можна описати наступним чином - стохастична динамічна задача розподілу запасів після катастроф (SDPDIAP) з невідомим попитом і постачанням. У багатоперіодичному контексті функції витрат на позбавлення (страждання від недостачі допомоги) часто дискретизуються на фіксовані часові кроки, що зберігає суперлінійний характер вихідної неперервної функції. Оскільки ми розглядаємо дискретні епохи прийняття рішень $t \in T = \{0,1...,T\}$, то також дискретизуємо нелінійні витрати на позбавлення для кожної епохи. Ми використовуємо ту ж функцію витрат на позбавлення $\gamma(\delta)$ від часу позбавлення $\delta$ (рівн 2.4).

$$\gamma(\delta) = e^{0.065\delta} - 1 \qquad (2.4)$$

Кількість осіб, для яких попит не задоволено, позначається як дефіцит $h_{tn}$. Витрати на позбавлення для кожної особи в районі множаться на дефіцит $h_{tn}$ для отримання витрат на позбавлення в районі.

Граничні витрати на позбавлення між двома епохами прийняття рішень визначаються як $g(\delta_{tn}) = \max\{0, \gamma(\delta_{tn}) - \gamma(\delta_{tn} - 1)\}$ Час і витрати на позбавлення зменшуються до нуля, коли прибувають нові запаси, що повністю

задовольняють попит на момент t, і знову зростають після вичерпання запасів. Якщо попит не задоволено, витрати на позбавлення розраховуються лише для дефіциту, але час позбавлення не скидається на нуль. Метою є мінімізація транспортних витрат та загальних витрат на позбавлення по всіх районах n∈N та епохах t∈T, що забезпечить розподіл ресурсів відповідно до потреб кожного району з використанням відповідного транспортного засобу.

Для реалізації покрокового розподілення ресурсів скористаємося процесом прийняття рішень Маркова (MDP). Разом з тим ми використаємо змінні стану, змінні рішення, екзогенну інформацію, функцію переходу, а також цільову функцію та оптимальну політику

**Змінні стану**

$S_t$ для $t \in T = \{1, 2, ..., T\}$

$$S_t = (I_t^{CW}, (I_{tn}, h_{tn}, \delta_{tn}, d_{t,t+1,n})_{n \in N}) \tag{2.5}$$

**Змінні рішення**

Матриця $x_t = [x_{tnk}]_{n \in N, k \in K}$ позначає рішення, що описує кількість ресурсів, виділених кожному району n за кожним транспортним режимом k у момент часу t. Сума розподілених ресурсів обмежена кількістю запасів, доступних у центральному складі (CW). Запаси допомоги не обов'язково мають бути повністю розподілені по районах; їх можна також залишити на складі для майбутніх розподілів. Множина можливих рішень у певному стані описується (рівн 2.6)

$$X(S_t) = \{x_t \in \mathbb{N}^{|K \| N|} : \sum_{k \in K} \sum_{n \in N} x_{tnk} \leq I_t^{CW}\} \tag{2.6}$$

**Екзогенна інформація**

Між послідовними моментами прийняття рішень t та t+1 надходить нова інформація. Реалізації випадкових змінних у новій інформації позначаються за допомогою оператора hat. Зовнішній інформаційний процес у нашій задачі включає надходження ресурсів у центральний склад $\hat{s}_{t+1}^{CW}$, а також для кожного району n∈N реалізацію потреб $\hat{d}_{t+1,n}$, протягом періоду [t, t+1) і прогнози потреб $d_{t+1,t+2,n}$ у наступному періоді [t+1,t+2), що повідомляються працівниками гуманітарних служб у районах. Нехай $\omega \in \Omega$ описує траєкторію зразка $W_1, W_2 ... W_T$ ,з певною реалізацією зовнішньої інформації $W_{t+1}$ визначеною як (рівн 2.7)

$$W_{t+1} = (\hat{s}_{t+1}^{CW}, (\hat{d}_{t+1,n}, d_{t+1,t+2,n})_{n \in N}) \tag{2.7}$$

**Функція переходу має наступний вигляд**:

$$I_{t+1}^{Cw} = I_t^{CW} - \sum_{k \in K} \sum_{n \in N} x_{tnk} + \hat{s}_{t+1}^{CW} \tag{2.8}$$

$$I_{t+1,n} = \max\{0, I_{tn} + \sum_{k \in K} x_{tnk} - \hat{d}_{t+1,n}\} \tag{2.9}$$

$$h_{t+1,n} = \max\{0, \hat{d}_{t+1,n} - I_{tn} + \sum_{k \in K} x_{tnk}\} \tag{2.10}$$

$$\delta_{t+1,n} = \begin{cases} \delta_{tn} + 1, I_{tn} + \sum_{k \in K} x_{tnk} \le \hat{d}_{t+1,n} \\ 0 \end{cases} \tag{2.11}$$

Інвентаризаційний статус центрального складу оновлюється відповідно до рішень про розподіл і надходження нових запасів (рівняння 2.8). У районах інвентаризація збільшується за рахунок розподілених ресурсів і зменшується через фактичний попит протягом періоду або стає нулем, якщо попит перевищує запаси разом із розподіленими ресурсами (рівняння 2.9). Дефіцит стає додатним, якщо попит перевищує запаси й розподіли (рівняння 2.10),

інакше дорівнює нулю. Час позбавлення збільшується на одиницю, якщо запаси після прийняття рішення недостатні для задоволення попиту, інакше стає нулем (рівняння 2.11). Останнє, нова оцінка попиту формується з екзогенної інформації.

**Цільова функція та оптимальна політика (прийняття рішень)**

Мета полягає в мінімізації загальних витрат на позбавлення та транспортних витрат упродовж часу для всіх районів. Витрати в епоху прийняття рішення t визначаються як (рівн 2.12)

$$C(S_t, x_t) = \sum_{n \in N} g(\delta_{tn}) h_{tn} + \sum_{k \in K} \sum_{n \in N} \left\lceil \frac{x_{tnk}}{q_k} \right\rceil c_{nk} \tag{2.12}$$

Перша частина враховує витрати від нестачі ресурсів $g(\delta)$, які залежать від часу нестачі $\delta_{tn}$ та дефіциту $h_{tn}$ у районах, а друга — транспортні витрати, що обчислюються як необхідна кількість транспортних засобів до кожного району. Ця кількість визначається розподіленими ресурсами $x_{tnk}$ та місткістю транспортного засобу $q_k$, помноженими на витрати $c_{nk}$ на одиницю транспортування. У кінцевому стані $S_T$ рішення не приймаються, а розраховуються завершальні витрати на нестачу за останній період [T−1,T) відповідно до функції витрат.

Нехай π — політика з множини політик П, а — функція прийняття рішень $X_t^{\pi}(S_t)$ для π, що визначає рішення $x_t \in X(S_t)$ для кожного стану $S_t \in S$. Мета — знайти оптимальну політику $\pi^*$, яка мінімізує очікувані витрати $C_t$ на всіх етапах прийняття рішень t∈T, починаючи з початкового стану $S_0$, відповідно до заданої цільової функції (рівн. 2.13).

$$\min_{\pi \in \Pi} \mathrm{E}\{\sum_{t=0}^{T} C(S_t, X_t^{\pi}(S_t)) \mid S_0\} \tag{2.13}$$

Обчислення оптимальної можна формально визначити, розв'язавши рівняння Беллмана. Для отримання рішення $x_t$ розглядаються безпосередні витрати $C(S_t, x_t)$ та очікувані майбутні витрати наступного стану $S_{t+1}$, щоб врахувати наслідки поточних рішень. Позначимо очікувані майбутні витрати стану $V_t(S_t)$ Оптимальні рішення $x_t^*$ на етапі t∈T можна знайти за (рівн 2.14)

$$x_t^* = \arg\min_{x_t \in X(S_t)} (C(S_t, x_t) + Expectation\{V_{t+1}(S_{t+1}) \mid S_t, x_t\} \quad (2.14)$$

$$S_{t+1} = S^M(S_t, x_t, W_{t+1}) \quad (2.15)$$

Розрахунок кращої політики у базовому вигляді вимагає великої обчислювальної потужності та багато часу. Для покращення цього процесу вдамося до апроксимації розрахунку без великих втрат у точності, використовуючи наступні методики:

Підхід наближеного динамічного програмування (ADP) з доповненням його методом декомпозиції. ADP є методом, що базується на симуляціях для навчання функції вартості (VFA), яка оцінює майбутні витрати, визначені значенням перебування в наступному стані $V_{t+1}(S_{t+1})$. Проблема вирішується шляхом ітеративного моделювання процесу Маркова (MDP) і навчання вартості перебування у певному стані на основі спостережуваних витрат.

Щоб уникнути перерахування простору результатів, ADP використовує концепцію пост-рішення стану. Стан пост-рішення $S_t^x$ — це стан системи одразу після прийняття рішення $x_t$ і до отримання нової інформації $W_{t+1}$ з $S_t^x = S^{Mx}(S_t, x_t)$, що описує детермінований перехід (рівн 2.16)

$$S_t^x = (I_t^{CW,x}, (I_{tn}^x, h_{tn}, \delta_{tn}, d_{t,t+1,n})_{n \in N}) \quad (2.16)$$

Де $I_t^{CW,x} = I_t^{CW} - \sum_{k \in K} \sum_{n \in N} x_{tnk}$, $I_{tn}^x = I_{tn} + \sum_{k \in K} x_{tnk}, \forall n \in N$

З пост-рішення стану $S_t^x$ ми можемо обчислити наступний стан $S_{t+1}$ за переходом $S_{t+1} = S^{MW}(S_t^x, W_{t+1})$, враховуючи зовнішню інформацію $W_{t+1}$. Обчислюючи майбутні витрати для пост-рішення стану $S_t^x$ замість наступного стану $S_t^x$ можна застосувати детерміновану функцію значення $V_t(S_t^x)$, замість очікуваного значення. $Expectation\{V_{t+1}(S_{t+1}) \mid S_t, x_t\}$, не використовуючи перерахування усіх можливих реалізацій поставок-запитів.

Через велику кількість можливих значень запасів і попиту, простір станів є дуже об'ємним, а додавання кожного нового району збільшує його виміри. Для скорочення обчислень використовують наближення функції значення (VFA), яке дозволяє уникнути повного перебору станів. Запропоновано підхід для вирішення проблеми - декомпозиційне лінійне наближення (DL-VFA), яке вивчає окремі лінійні функції для кожного району для забезпечення масштабованості.

На додаток, замість того, щоб наближати майбутні витрати стану за допомогою однієї функції значення, коли структура задачі дозволяє, її можна розкласти на кілька наближень. Розкладаємо по принципу – одна функція значення на одну точку видачі, а загальна функція отримується як їх сума. Наша модель класифікується як та, яка має кінцевий горизонт T (термін у MDP) і не стаціонарна, тому майбутні витрати залежні від часу і визначаються кожну епоху t. З цього можна зробити твердження, що функції значення (value functions) обчислюються не лише для кожної точки видачі окремо, а ще й окремо для кожної епохи $t \in T$. В результаті ми отримуємо декомпозовану лінійну апроксимацію функції вартості DL-VFA (рівн. 2.17)

$$V_t(S_t^x) \approx \bar{V}_t^{linear}(S_t^x) = \sum_{n \in N} \bar{V}_{tn}^{linear}(S_{tn}^x) = \sum_{n \in N} \Theta_{tn} \Phi^{linear}(S_{tn}^x), \forall t \in T \qquad (2.17)$$

Для покращення наближення функції значення в нашій цілісній функції, ми враховуємо нелінійність витрат через «позбавлення» або ж страждання через недостачу допомоги (deprivation).

Головні ознаки для векторів ознак $\Phi^{linear}$ це очікувана кількість позбавлень (deprivation cost) $\bar{G}^{x}_{t,t+1,n}$. Щоб отримати ці значення нам необхідно визначити очікуваний дефіцит $\bar{h}_{t,t+1,n}$, як кількість людей, які не отримали допомоги. Очікувана кількість позбавлень це добуток очікуваного дефіциту та коефіцієнту позбавлення $g(\delta_{tn}+1)$ (визначеному раніше) в наступному періоді. Так як попит не визначений додаймо 2 СКВ до визначення очікуваного дефіциту, отримаємо рівняння 2.19.

$$\bar{h}_{t,t+1,n} = \max\{0, d_{t,t+1,n} + 2\sigma_{d_{t,t+1,n}} - I^{x}_{tn}\} \tag{2.19}$$

Отже отримуємо очікувану кількість позбавлень (рівн 2.20)

$$\bar{G}^{x}_{t,t+1,n} = g(\delta_{tn}+1) * \bar{h}_{t,t+1,n} \tag{2.20}$$

Також важливим фактором є розташування точок видачі $I^{x}_{tn}$ та час протягом якого в них є позбавлення $\delta_{tn}$. Із використанням нейронних мереж нам необхідно враховувати також індекс часу t (індикатор епохи) та інвентар розподільного центру $I^{CW,x}_{tn}$. Тобто отримуємо наступні набори ознак (рівн 2.21):

$$\Phi^{linear}(S^{x}_{tn}) = (I^{x}_{tn}, \delta_{tn}, \bar{G}^{x}_{t,t+1,n}) \tag{2.21}$$

Мета DL-VFA на кожному етапі прийняття рішення включає прямі витрати на депривацію та транспортування, а також апроксимовані майбутні витрати для кожного району(рівн 2.22):

$$\min_{x_{tnk}\in X}\sum_{n\in N}\left(g(\delta_{tn})\hat{d}_{tn}+\sum_{k\in K}\left\lceil\frac{x_{tnk}}{q_k}\right\rceil c_{nk}+\Theta_{tn}\Phi^{linear}(S_{tn}^{x})\right) \tag{2.22}$$

Де $\Theta_{tn}$ - ваги, які навчаються

Модель має наступний набір обмежень

$$I_t^{CW,x}=I_{t-1}^{CW}-\sum_{k\in K}\sum_{n\in N}x_{tnk} \tag{2.23}$$

$$I_{tn}^{x}=I_{t-1,n}+\sum_{k\in K}x_{tnk} \tag{2.24}$$

$$h_{t,t+1,n}\geq d_{t,t+1,n}+\xi_{t,t+1,n}-I_{tn} \tag{2.25}$$

$$h_{t,t+1,n}\leq d_{t,t+1,n}+\xi_{t,t+1,n}-I_{tn}-M_n^h*(1-z_n^h) \tag{2.26}$$

$$h_{t,t+1,n}\leq M_n^h z_n^h \tag{2.27}$$

$$I_t^{CW},x_{tnk},I_{tn},h_{tn},\bar{G}_{tn}^{x},\bar{h}_{t,t+1,n}\geq 0 \tag{2.28}$$

$$z_n^h\in\{0,1\} \tag{2.29}$$

## 2.8. Опис алгоритму

Алгоритм навчання функції наближення значень у DL-VFA на основі наближеного динамічного програмування (ADP):

**Початкові оцінки функцій значень:**

- Замість випадкових початкових значень, які можуть ускладнити збіжність, ініціюємо функції значень з обґрунтованими оцінками.

- Генеруємо великий набір траєкторій з пострішень та витрат, використовуючи підготовчий евристичний алгоритм, і застосовуємо лінійну регресію для первісної оцінки функцій.
- Зберігаємо ці траєкторії в буфері початкового досвіду $B_0$.

**Епізодичне навчання з оновленням функцій значень:**

- Проводимо R епізодів навчання, де кожен епізод триває один горизонт.
- На кожній ітерації t вирішуємо задачу прийняття рішення для одного етапу, використовуючи з певною ймовірністю $\epsilon^r$ підхід із початкової евристики для уникнення локальних мінімумів. $\epsilon^r$ зменшується після кожного оновлення функцій.
- Обчислюємо пострішення та прямі витрати, які зберігаються в буфері досвіду .

**Оновлення ваг функцій значень:**

- Кожні u епізодів оновлюємо ваги функцій значень на основі буфера $\mathrm{B}^r$, що містить набір спостережень.
- Видаляємо вибірки, витрати яких перевищують 1.5 інтерквартильного розмаху над третім квартилем Q3+1.5IQR, для усунення впливу аномалій.
- Оцінюємо проміжні ваги функцій значень через метод найменших квадратів, згладжуючи старі ваги за допомогою проміжних із коефіцієнтом згладжування $\alpha^r$.
- Після кожного оновлення зменшуємо $\epsilon^r$ і $\alpha^r$

# 3. ПРОГРАМНА РЕАЛІЗАЦІЯ

## 3.1. Підсистема моніторингу

Детальної інформації про підсистему моніторингу у цьому розділі розкривати не доцільно. Бо технологія, розроблювана в цій роботі фокусується на оптимізації процесів розподілення гуманітарної допомоги. А результати опитувань використовує лише для визначення початкових даних. Тому достатнім буде коротко перелічити технології задіяні при розробці цього модуля.

Інформаційна система опитувань реалізована із застосуванням Spring Boot, який забезпечує створення RESTful API для взаємодії з клієнтською частиною, розробленою за допомогою React JS. Цей підхід дозволяє ефективно обробляти запити від користувачів і виконувати операції відповідно до ролей, зокрема адміністративної. Spring Boot спрощує розробку за рахунок попередньо налаштованих залежностей, таких як Spring Core, Spring Web MVC, та Jackson Databind, що сприяє інтеграції функціоналу та забезпечує зручність реалізації архітектури Model-View-Controller.

**Функціонал клієнтської та серверної частин**

Клієнтська частина, створена на основі React JS із Redux і Redux Thunk для централізованого збереження стану компонентів, слугує інтерфейсом для авторизованих користувачів, зокрема адміністраторів, які можуть використовувати всі можливості додатку.. Redux дозволяє обробляти стан додатку через редуктори і дії, які також відправляють запити до серверу. Для роботи з картами застосовується бібліотека react-google-maps/api, а для оптимізації відображення використовується кластеризація точок за допомогою useSupercluster. Ці технології забезпечують модульність, надійність і масштабованість інформаційної системи Взаємодія клієнт-сервер організована через REST API, запити якого обробляються сервером, виконуючи необхідну бізнес-логіку. Окремі сервіси на серверній стороні забезпечують обробку

даних, доступ до баз даних або взаємодію із зовнішніми API, наприклад Telegram API, для організації опитувань серед користувачів.

**Telegram-бот і його інтеграція**

Найшвидший спосіб реалізації опитувань – використання Telegram Bot API. . Його основна функція — збір геолокаційних даних від користувачів без додаткових особистих даних. Бот автоматично визначає координати користувача, його місце перебування, та асоціює його з відповідним населеним пунктом. Надалі, якщо проводиться опитування за визначеним містом, користувач отримує повідомлення для уточнення, чи потребує він гуманітарної допомоги. Якщо користувач бере участь в опитуванні, йому надсилається інформація про найближчі точки отримання допомоги, коли вони визначені. Завдяки бібліотеці Aiogram, написаній на Python, яка має багато документації, розробка значно спрощується. Однак, Telegram Bot API має обмеження, тому для майбутнього розвитку і розширення системи рекомендується створити мобільний додаток. Для роботи бота використовуються команди, прив’язані до функцій, і станів користувачів через механізм скінченних автоматів. Це дозволяє поетапно обробляти дії користувача: отримання місця розташування, підтвердження даних і участь в опитуваннях. Стан користувачів зберігається в MongoDB, що забезпечує надійність даних. Для оберненого геокодування використовується бібліотека Geopy, а для роботи з PostgreSQL – psycopg2 із захистом від SQL-ін’єкцій.

**Використання Spring JPA та Hibernate**

Для взаємодії серверної частини із реляційною базою даних застосовується Spring JPA на основі Hibernate, що забезпечує роботу з даними через ORM (Object-Relational Mapping). Це значно спрощує управління базою даних, дозволяючи автоматично створювати запити за назвами методів у репозиторіях. Наприклад, можна виконувати фільтрацію даних або використовувати мову JPQL для складніших запитів. Створення сутностей і їхніх зв’язків між таблицями виконується з використанням анотацій, таких як

@Entity, @ManyToOne, і @Embedded. Це полегшує структурування даних та автоматизує роботу з ними.

**Архітектура контролерів і бізнес-логіки**

Архітектура серверної частини побудована за шаблоном Model-View-Controller (MVC). Контролери забезпечують обробку HTTP-запитів (GET, POST, PUT, PATCH, DELETE) і реалізацію RESTful принципів. Наприклад, ендпоінт POST /admin/addCity дозволяє адміністраторам додавати нові міста, включаючи їх назви, координати та іншу інформацію. Запити валідуються, і у випадку помилок сервер повертає структуровані повідомлення у форматі JSON. Це забезпечує захищеність даних і точність їхнього внесення.

Бізнес-логіка, реалізована у вигляді сервісних класів, виступає проміжним шаром між контролерами та репозиторіями. Вона дозволяє виконувати як прості операції, так і складніші процеси, наприклад, організацію опитувань або обробку великих обсягів даних. Це гарантує розширюваність системи та її здатність підтримувати нові функції.

Загальну архітектуру моніторингу можна побачити на рис 3.2

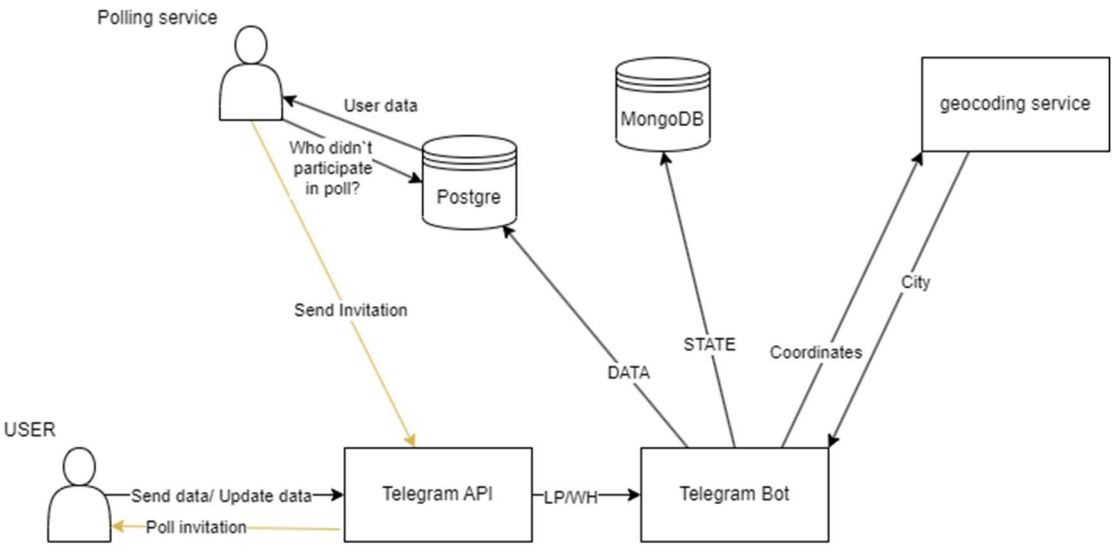


Рисунок 3.1- архітектура модуля опитування [22]

Результати роботи моніторингу (рис 3.2)

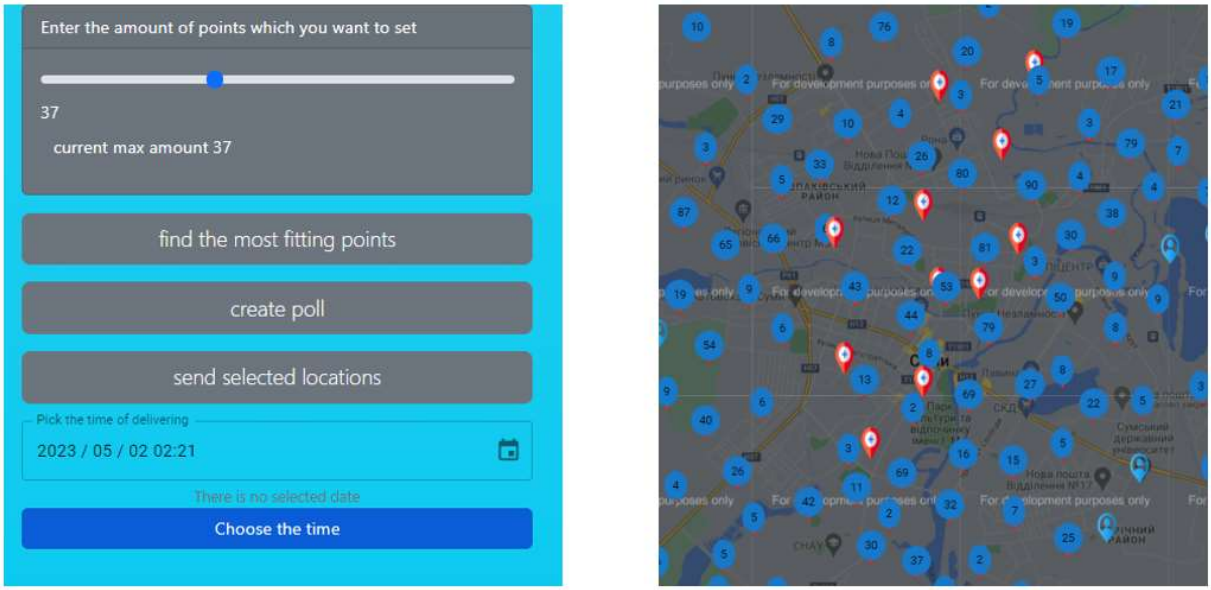


Рисунок 3.2 – результат роботи модуля моніторингу[22]

## 3.2. Оптимізація процесів розподілення та архітектура системи

### Архітектура системи

На рисунку 3.3 можна побачити основну архітектуру, закладену при вирішенні задачі оптимізації розподілення гуманітарної допомоги. Як можна помітити, всі компоненти задовольняють умови, які ми окреслили у розділі 2. А саме надходження нової інформації від призначених осіб, яка надходить в певний проміжок T з використанням розподіленого сховища подій apache Kafka. Вирішення транспортної задачі MIP для знаходження оптимального розподілу вантажів. Використання та навчання апроксимованих функцій для прийняття рішень в динамічних умовах. За рисунком можна також визначити, що початкові умови залежать від результатів моніторингу, що надає можливість створити зв’язок між двома підсистемами. Початкові умови використовуються у евристичному початковому алгоритмі для уникнення знаходження локальних мінімумів та в процесі оптимізації апроксимованих функцій витрат, шляхом модифікації ваг лінійних функцій для кожної точки видачі.

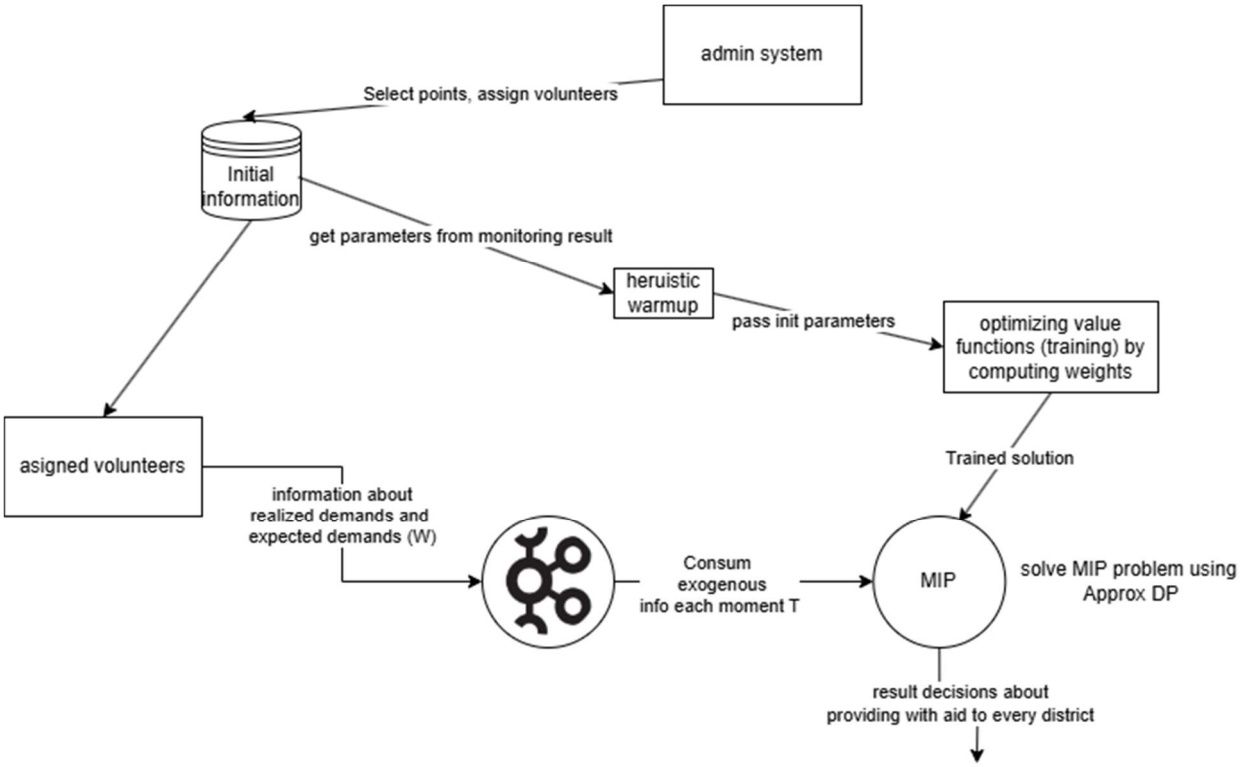


Рисунок 3.3 – архітектура технології оптимізації розподілу вантажів у динамічних умовах

Реалізація алгоритму

Розглянемо детальніше кожен крок алгоритму оптимізації функції.

Евристичний алгоритм для створення початкового буферу B

```
def warmup(max_expected_demands_inc,
           q,
           transport_cost,
            n = 4,
           init_supply=abs(int(np.random.normal(loc=3400, scale=4))),
           max_supply_inc=abs(int(np.random.normal(100, scale=4))),
           buffer_size = 1000,
           T = 6,
           discount_factor=0.9):
    BufferSize = buffer_size
    Buffer = []

    for i in range(BufferSize):
        actual_supply = init_supply
        W0, W =
```

```
generate_path(actual_supply,max_expected_demands_inc,max_supply_inc,amount_of
_districts=n, epoch_time_amount=T)
        # warehouse, inv, shortage (amount of unsatisfied users after
distribution in moment t), time of deprivation, expected demand
        S = [(W0[0], (np.zeros(n, dtype=int), np.zeros(n, dtype=int),
np.zeros(n, dtype=int), W0[1][1]))]
        S, xtnk, trajectory_buffer = define_x_heruistic(S, W, T, n,
transport_cost, q)
        Buffer.append(trajectory_buffer)
    Buffer = remove_outliers(Buffer)
    # remove outliers
    return Buffer
```

Визначення рішень, використовуючи евристичний алгоритм мовою Python

```
def define_x_heruistic(S,W, T,n,transport_cost, q ):
    xtnk = np.zeros((T, n, 2), dtype=int)
    Icwx = np.zeros(T, dtype=int)
    Sx = []
    trajectory_buffer = []
    for j in range(T):
        Icws, (It, ht, delt, dt) = S[j]
        # obtain a decision xt
        Icwx[j] = Icws
        for k in range(n):
            # print(S[j][1][2])
            if (S[j][1][2][k] >= np.random.randint(low=1, high=3)):

                xinterim = q[0][1] * np.random.randint(low=1, high=3)

                if (xinterim <= Icwx[j]):
                    xtnk[j][k][0] = xinterim
                    Icwx[j] -= xinterim

        n1 = np.random.randint(low=0, high=n)

        xtemp = np.minimum(q[1][1], Icwx[j] + xtnk[j][n1][0])

        if (S[j][1][2][n1] >= np.random.randint(low=1, high=3)):
            xtnk[j][n1][0] = 0
            xtnk[j][n1][1] = xtemp
```

```
        Sx.append(
            (S[j][0] - np.sum(xtnk[j]), (S[j][1][0] + np.sum(xtnk[j,:],
axis=1), S[j][1][1], S[j][1][2], S[j][1][3])))
        Icwxs, (Itnx, htnx, deltax, dtx) = Sx[-1]
        # arrival of supply, realization of demand, expected demand+--
        sCW, (dh, dt1) = W[j]
```

Тут обирається розподіл вантажів навмання, підпорядковуючись законам рівномірного розподілу на проміжку unif [1, 3] для кожного пункту видачі, де час очікування delt >= unif[1,3]

Генерація екзогенної інформації відбувається у функції generate_path. Для її реалізації використовуються різноманітні генератори випадкових чисел основою для яких є базові метрики, отримані в результаті процесу моніторингу.

Функцію вартості можна представити у наступному вигляді.

```
def cost_funct(St: tuple, Xt: np.array, transport_costs,
vehicles_characteristics):
    Iwh, (Inv, shortage, time_of_depr, expected_demand) = St

    vehicles_transport_cost = vehicles_characteristics[:,0] * np.transpose(
        np.array([transport_costs, ] * vehicles_characteristics[:, 0].size))
    res = 0
    for i in range(Xt.shape[0]):
        # print(Xt.shape[0])
        interm = np.zeros(2, dtype=int)
        interm = np.add(Xt[i], interm)
        res += np.sum(np.sum(np.ceil(np.divide(interm,
vehicles_characteristics[:, 1])) * vehicles_transport_cost[i]))
    depr_sum = 0
    for r in range(time_of_depr.size):
        depr_sum += deprivation_cost(time_of_depr[r])*shortage[r]
    return depr_sum + res
```

Цю функцію можна інтерпретувати як балансування між дороговизною доставки та часом очікування деякою кількістю потребуючих допомоги.

Для процесу пошуку коефіцієнтів функції отриманої в результаті побудови лінійної регресії на за методом найменших квадратів, використаємо бібліотеку sklearn:

```
def compute_weights(features, y):
    reg = [[sklearn.linear_model.LinearRegression().fit(features[:, t, :, n],
[yl[n] for yl in y[:, t]]) for n in
            range(len(y[:, t][0]))] for t in range(y.shape[1])]
    weights_coef = np.zeros((len(reg), len(reg[0]), 4))
    for tidx, t in enumerate(reg):
        # reg1.score(features, y)
        for nidx, regr in enumerate(reg[tidx]):
            intercept = regr.intercept_
            for l in range(len( regr.coef_)):
                weights_coef[tidx][nidx][l] = regr.coef_[l]
            weights_coef[tidx][nidx][-1] = intercept
    return weights_coef
```

Де набір ознак складається з тих, що були визначені у рівнянні (2.21)

Сам процес оптимізації опісля використання евристичного алгоритму проводиться з експлуатацією алгоритму лінійного програмування MIP, що комбінує набори цілочисельних змінних разом з неперервними змінними. В нашому випадку неперервні – значення позбавлень від кількості нерозподіленої за попитом допомоги та часу, протягом якого ця допомога не надавалася, а цілочисельні змінні – рішення щодо розподілу цієї допомоги у кожному пункті видачі. Але найбільш важливою відмінністю від звичайних алгоритмів динамічного програмування для вирішення подібних задач (де просто перераховуються усі комбінації рішень і вибирається оптимальне, що може бути, як мінімум, невигідним з точки зору використання обчислювальних ресурсів, а то й неможливим в умовах нестаціонарних або ж динамічних систем) є використання апроксимованої функції майбутніх витрат в цільовій функції.

Для вирішення проблеми MIP використаємо бібліотеку gurobipy

Ініціалізуємо змінні моделі

```
env = Env()
env.setParam('OutputFlag', 0)
Icws, (It, ht, delt, dt) = S[j]
MAX_P = 200000
# Initialize model
model = Model("HumanitarianOptimization")
model.Params.IntFeasTol = 1e-9
q_gb = np.copy(q)
# I
# Decision variables
K = q_gb.shape[0]
x = model.addVars(n, K, name="x", vtype=GRB.INTEGER, lb=0)
I_CW = model.addVar(name="I_CW", vtype=GRB.INTEGER, lb=0)
I = model.addVars(n, name="I", vtype=GRB.INTEGER, lb=0)
h = model.addVars(n, name="h", vtype=GRB.CONTINUOUS, lb=0)
G_x = model.addVars(n, name="G_x", vtype=GRB.CONTINUOUS, lb=0)
z_h = model.addVars(n, name="z_h", vtype=GRB.BINARY)
```

Додаємо обмеження

```
ceilparam = model.addVars(n, K, vtype=GRB.INTEGER, lb=0)
model.addConstrs(x[ni, k] >= 0 for ni in range(n) for k in range(K))
model.addConstrs(h[ni] >= 0 for ni in range(n))
model.addConstrs(I[ni] >= 0 for ni in range(n))
model.addConstrs(G_x[ni] >= 0 for ni in range(n))
model.addConstr(I_CW >= 0)

model.addConstr(
    I_CW == Icws - quicksum(x[ni, k] for ni in range(n) for k in range(K)),
    name="InventoryBalanceCW"
)
model.addConstrs(
    (I[ni] == It[ni] + quicksum(x[ni, k] for k in range(K)) for ni in
range(n)),
    name="InventoryBalanceItem"
)

# Auxiliary variable
model.addConstrs(
```

```
    (G_x[ni] == deprivation_cost(delt[ni] + 1) * h[ni] for ni in range(n)),
    name="AuxiliaryGx"
)
model.addConstrs(
    ceilparam[ni, k] >= x[ni, k] / q_gb[k][1] for ni in range(n) for k in
range(K)
)
model.addConstrs(
    ceilparam[ni, k] <= x[ni, k] / q_gb[k][1] + 0.999 for ni in range(n) for
k in range(K)
)
# Conditional constraints
model.addConstrs(
    (h[ni] >= dt[ni] + 2 * std_vals[j][ni] - I[ni] for ni in range(n)),
    name="CondConstraint1"
)
model.addConstrs(
    (h[ni] <= dt[ni] + 2 * std_vals[j][ni] - I[ni] - MAX_P * (1 - z_h[ni])
for ni in range(n)),
    name="CondConstraint2"
)
model.addConstrs(
    (h[ni] <= MAX_P * z_h[ni] for ni in range(n)),
    name="CondConstraint3"
)
```

Визначаємо цільову функцію

```
weighted_t = weights_coef[j]
model.setObjective(
    quicksum(deprivation_cost(delt[ni]) * ht[ni] for ni in range(n)) +
    quicksum(ceilparam[ni, k]*transport_cost[ni]  for ni in range(n) for k in
range(K))+
    quicksum(weighted_t[ni][0] * I[ni] + weighted_t[ni][3] +
weighted_t[ni][1]*delt[ni]+weighted_t[ni][2]*G_x[ni] for ni in range(n)),
    GRB.MINIMIZE
)

model.optimize()
```

Усі обмеження відповідають формулам (2.22-2.29)

Проаналізуємо результати для м. Суми За декількома сценаріями

1й сценарій – має n=3 кількість центрів розподілу і T=8 епізодів (наприклад інформація подається щогодини)

Таблиця 3.1 – Розподіл вантажів за результатами моніторингу у м. Суми(1)

| t | ICW | $It_1$ | $It_2$ | $It_3$ | $X_1$ | $X_2$ | $X_3$ | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 2825 | 0 | 0 | 175 | 0 | 0 | 175 | 18 | 0 | 1 |
| 3 | 2825 | 0 | 0 | 96 | 0 | 0 | 0 | 64 | 16 | 0 |
| 4 | 2825 | 0 | 0 | 89 | 0 | 0 | 0 | 80 | 37 | 0 |
| 5 | 2724 | 101 | 0 | 36 | 101 | 0 | 0 | 108 | 38 | 0 |
| 6 | 2724 | 0 | 0 | 22 | 0 | 0 | 0 | 50 | 48 | 0 |
| 7 | 2604 | 120 | 0 | 9 | 120 | 0 | 0 | 56 | 49 | 0 |
| 8 | 2604 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 50 | 2 |

За таблицею 3.1 видно очевидний недолік – іноді, послідовність рішень така, що не береться до уваги деякий з центрів розподілення через досить малий приріст попиту, та це можна виправити, граючись з критеріями оцінювання показника позбавлень.

Таблиця 3.2 - Розподіл вантажів за результатами моніторингу у м. Суми(2)

| t | ICW | $It_1$ | $It_2$ | $It_3$ | $X_1$ | $X_2$ | $X_3$ | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 3000 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 1 | 36 |
| 3 | 3000 | 0 | 0 | 0 | 0 | 0 | 0 | 24 | 34 | 56 |
| 4 | 3000 | 0 | 0 | 0 | 0 | 0 | 0 | 47 | 105 | 86 |
| 5 | 2896 | 0 | 0 | 104 | 0 | 0 | 104 | 60 | 121 | 110 |
| 6 | 2826 | 70 | 0 | 0 | 70 | 0 | 0 | 68 | 167 | 56 |
| 7 | 2711 | 0 | 115 | 0 | 0 | 115 | 0 | 23 | 192 | 56 |
| 8 | 2567 | 0 | 154 | 0 | 0 | 154 | 0 | 64 | 85 | 70 |

В цьому випадку (таблиця 3.2) – розподіл більш якісний, проте потребуючим, які прийшли раніше за інших доведеться чекати.

В цілому, аналізуючи результати роботи програми, можна сказати, що використання наближеного динамічного програмування для вирішення задач оптимізації розподілу гуманітарної допомоги є непоганим рішенням, якщо існує попередня інформація для проведення навчання - саме для цього ми використовуємо модуль моніторингу потреб. При взаємодії два методи дають непогані результати. В чому і є основна новизна цієї роботи. Але варто пам’ятати, що це лише наближені результати і, трапляється, що оптимізація працює не в тому напрямку, тому цю систему можна використовувати як допоміжну при прийнятті рішень та ретельно перевіряти результати.

# ВИСНОВОК

В результаті огляду наявних рішень та методів вирішення задач оптимізації розподілу вантажів було виявлено, що наявні на 2024 рік системи в Україні не мають схожого функціоналу з тим, який надає ця технологія. Також було виявлено підхід і конкретну його реалізацію для вирішення наших задач – вирішення задачі наближеного динамічного програмування шляхом розподіленої лінійної апроксимації функцій витрат або ж DL-VFA [5].

Також було проаналізовано основні технології як для створення підсистеми моніторингу і технології оптимізації розподілення вантажів. Для створення підсистеми моніторингу було обрано Java, Spring Boot для серверної частини, Telegram API для системи опитувань, ReactJS – для клієнтської частини. Для розробки технології оптимізації розподілення гуманітарної допомоги було обрано мову програмування Python через його велику вживаність і доцільність при аналізі великих об’ємів даних із супутніми бібліотеками: sklearn, numpy, gurobipy та ін. В якості способу обробки і контролю надходження нової інформації щодо попиту було обрано Telegram API та розподілене сховище подій Apache Kafka.

Розроблено чітку архітектуру додатку, який складається з двох підсистем, які взаємодіють між собою (і кожної підсистеми).

Було продемонстровано модулі підсистеми моніторингу потреб і додано ілюстрацію взаємодії її компонентів.

Визначено і впровадження основні кроки алгоритму оптимізації розподілення допомоги з використанням наближеної функції витрат, створено апроксимацію функції, використовуючи лінійну регресію.

Модуль моніторингу має клієнтську частину і гарно відображає результати опитувань. Натомість компонент оптимізації повертає дані у певному форматі, який згодом можна легко відобразити, підпорядковуючись архітектурному стилю REST та Javascript бібліотеки/фреймворки.

Для демонстрації роботи було надано ER-діаграми та різні схеми взаємодії компонентів систем.

В результаті розроблено систему, яка об’єднує моніторинг потреб і оптимізацію розподілення гуманітарних вантажів, ґрунтуючись на результатах опитувань, проте не оцінюючи інформацію з моніторингу як 100% достовірну, що є досить вдалим рішенням, адже система не вимагає надання вразливої інформації у користувачів для підтвердження ідентифікації, тому є неточною. В чому і полягає унікальність системи.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Кондрашова, С.С. Інформаційні технології в управлінні [Текст] : Навч. посібник / С.С.Кондрашова . — К. : МАУП, 1998. — 560 с.
2. Гуманітарна катастрофа чи гуманітарна голка – дві сторони однієї медалі: доступ до гуманітарної допомоги в умовах збройного конфлікту на сході України / О.А. Біда, А.Б. Блага, О.А. Мартиненко, М. Г. Статкевич; за заг. ред. А.П. Бущенка / Українська Гельсінська спілка з прав людини. – К., КИТ, 2016. – 54 р. – с. 4 [Електронний ресурс] / Режим доступу: https://helsinki.org.ua/wpcontent/uploads/2016/06/Press_Humanitarian_aid_reportUKR.pdf
3. Механізми надання державою гуманітарної допомоги в умовах воєнного стану / С. Деркач та ін. Київ : Нац. агенство з питань запобігання корупції, 2022. 75 с. URL: https://nazk.gov.ua/wp-content/uploads/2022/07/Mehanizmy-nadannya-derzhavnoyu-gumanitarnoyi-dopomogy-v-umovah-voyennogo-stanu.pdf (дата звернення: 13.11.2024).
4. Оксана Стельмах. U-LEAD with Europe - Зміни у законодавстві про гуманітарну допомогу: кого стосується і як діяти... [Електронний ресурс]. 2024. Режим доступу: https://u-lead.org.ua/news/388 (accessed: 13.11.2024).
5. van Steenbergen R., van Heeswijk W., Mes M. The Stochastic Dynamic Post-Disaster Inventory Allocation Problem with Trucks and UAVs. 2023. URL: https://arxiv.org/abs/2312.00140
6. Swasdee A., Anshari M., Hamdan M. Artificial intelligence as decision aid in humanitarian response. 2020 international conference on decision aid sciences and application (DASA). 2020. С. 773–777. URL: https://doi.org/10.1109/DASA51403.2020.9317111.
7. A decision-making model to determine dynamic facility locations for a disaster logistic planning problem using deep learning / L. Tanti та ін. *Algorithms*. 2023. Т. 16, № 10. URL: https://doi.org/10.3390/a16100468.

8. Волошин О. Як обрати архітектуру для веб додатку [Електронний ресурс] / Олексій Волошин // hillel blog. – 2022. – Режим доступу до ресурсу: https://blog.ithillel.ua/articles/web-application-architecture.
9. TIOBE Index - TIOBE [Electronic resource]. URL: https://www.tiobe.com/tiobe-index/ (accessed: 10.11.2024).
10. Wozniewicz B. The Difference Between a Framework and a Library [Електронний ресурс] / Brandon Wozniewicz. – 2019. – Режим доступу до ресурсу: https://www.freecodecamp.org/news/the-difference-between-a-framework-and-a-library-bd133054023f/.
11. Silnitsky N. How to choose the right database for your service [Електронний ресурс] / Natan Silnitsky. – 2021. – Режим доступу до ресурсу: https://medium.com/wix-engineering/how-to-choose-the-right-database-for-your-service-97b1670c5632.
12. Dharmendra K. Best Databases To Use In 2022 [Електронний ресурс] / Kumar Dharmendra. – 2022. – Режим доступу до ресурсу: https://hevodata.com/learn/best-database/.
13. Calculate distance, bearing and more between Latitude/Longitude points [Електронний ресурс] // Movable Type Scripts – Режим доступу до ресурсу: https://www.movable-type.co.uk/scripts/latlong.html.
14. Lai J. Z., Huang T.-J., Liaw Y.-C. A fast k-means clustering algorithm using cluster center displacement. Pattern recognition. 2009. Т. 42, № 11. С. 2551-2556. URL: https://doi.org/10.1016/j.patcog.2009.02.014.
15. Arthur D., Vassilvitskii S. K-Means++: the advantages of careful seeding. Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms. USA, 2007. С. 1027–1035.
16. Schubert E., Rousseeuw P. J. Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms. Similarity search and applications / ред.: G. Amato та ін. Cham, 2019. С. 171--187.
17. Clustering large applications (program CLARA). Finding groups in data. 1990. С. 126-163. URL: https://doi.org/10.1002/9780470316801.ch3.

18. Spath P., Spath P. Building Single-Page Web Applications with REST and JSON. *Beginning Jakarta EE: Enterprise Edition for Java: From Novice to Professional*. 2019. С. 133–163.
19. REST: From Research to Practice - Google книги: / за ред. Erik Wilde, Cesare Pautasso. London: Springer Science, 2011. 21–22с.
20. Jones M. JSON Web Token (JWT) [Електронний ресурс] / M. Jones, J. Bradley, N. Sakimura // Internet Engineering Task Force. – 2015. – Режим доступу до ресурсу: https://www.rfc-editor.org/rfc/rfc7519
21. Balcik B., Beamon B., Smilowitz K. Last mile distribution in humanitarian relief. *Journal of intelligent transportation systems*. 2008. Т. 1818. URL: https://doi.org/10.1080/15472450802023329.
22. Ільченко Є. В. Інформаційна онлайн-система моніторингу гуманітарних потреб населення : робота на здобуття кваліфікаційного ступеня бакалавра : спец. 122 - комп'ютерні науки / наук. кер. Б. О. Кузіков. Суми : Сумський державний університет, 2023. 73 с.

# ДОДАТОК А

Algo.py

```
import math
import random

import gurobipy
import numpy
import statistics
import numpy as np
import pandas as pd

import itertools
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import matplotlib
import numpy as np
import sys
from sklearn import *
import sklearn.preprocessing

from gurobipy import *

from numpy.random import default_rng


# This is a sample Python script.

# Press Shift+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool windows,
actions, and settings.


def print_hi(name):
    # Use a breakpoint in the code line below to debug your script.
    print(f'Hi, {name}')  # Press Ctrl+F8 to toggle the breakpoint.


def deprivation_cost(time: int):
    return math.exp(0.065 * time) - 1


# St: warehouse, inv, shortage (amount of unsatisfied users after
distribution in moment t), time of deprivation, expected demand
# Xt: 3d array of supplyment
def cost_funct(St: tuple, Xt: np.array, transport_costs,
vehicles_characteristics):
    Iwh, (Inv, shortage, time_of_depr, expected_demand) = St

    vehicles_transport_cost = vehicles_characteristics[:,0] * np.transpose(
        np.array([transport_costs, ] * vehicles_characteristics[:, 0].size))
    res = 0
    for i in range(Xt.shape[0]):
        # print(Xt.shape[0])
        interm = np.zeros(2, dtype=int)
        interm = np.add(Xt[i], interm)
        res += np.sum(np.sum(np.ceil(np.divide(interm,
vehicles_characteristics[:, 1])) * vehicles_transport_cost[i]))
    depr_sum = 0
    for r in range(time_of_depr.size):
        depr_sum += deprivation_cost(time_of_depr[r])*shortage[r]
```

```
    return depr_sum + res


def cost_funct_n(St: tuple, Xt: np.array, transport_costs,
vehicles_characteristics):
    ICW, (Ic, sh, delt, d) = St
    N = Ic.size
    realized_costs = np.zeros( N)
    vehicles_transport_cost = vehicles_characteristics[:, 0] * np.transpose(
        np.array([transport_costs, ] * vehicles_characteristics[:, 0].size))
    for n in range(N):

        res = 0
        for it, xtn in enumerate(Xt[n]):
            res = xtn / vehicles_characteristics[:, 1][it]
            res = math.ceil(res) * vehicles_transport_cost[n][it]
        depr_sum = deprivation_cost(delt[n]) * sh[n]
        res += depr_sum
        realized_costs[n] = res
    return realized_costs

def learnFromFeatures(buffer: []):
    return

def compute_realized_future_costs(St, Xt: np.array, transport_costs,
vehicles_characteristics, lambda_discount):
    """
    Recursively computes realized future district costs for all time steps
and districts.

    Args:
        buffer: List of experiences [(state, decision, cost,
post_decision_state)].
        lambda_discount: Discount factor λ.

    Returns:
        realized_costs: Dictionary of realized future costs V^tn for each t ∈
T, n ∈ N.
    """
    ICW = np.array([state[0] for state in St])  # Перший елемент кожного
запису
    Ic = np.array([state[1][0] for state in St])  # Перший елемент всередині
кортежа
    sh = np.array([state[1][1] for state in St])  # Другий елемент
    delt = np.array([state[1][2] for state in St])  # Третій елемент
    d = np.array([state[1][3] for state in St])
    T = len(ICW)
    N = Ic[0].size

    realized_costs = np.zeros((T, N))
    vehicles_transport_cost = vehicles_characteristics[:, 0] * np.transpose(
        np.array([transport_costs, ] * vehicles_characteristics[:, 0].size))

    for t in range(T - 2, -1, -1):  # Start from the last time step and move
backward
        for n in range(N):

            res = 0

            for it,xtn in enumerate( Xt[t][n]):
                res = xtn/vehicles_characteristics[:,1][it]
                res = math.ceil(res)*vehicles_transport_cost[n][it]
```

```
            depr_sum = deprivation_cost(delt[t][n]) * sh[t][n]

            res+= depr_sum
            if t == T - 1:
                realized_costs[t][n] = res
            else:
                # Compute the discounted future costs recursively
                realized_costs[t][n] = res + lambda_discount *
realized_costs[t + 1][n]

    return realized_costs

# def mip_definer:

def compute_realized_future_costs_using_cost(cost, lambda_discount):
    T = len(cost)
    N = len(cost[0])
    realized_costs = np.zeros((T, N))

    for t in range(T - 2, -1, -1):  # Start from the last time step and move
backward
        for n in range(N):
            res = 0
            depr_sum = cost[t][n]
            res += depr_sum
            if t == T - 1:
                realized_costs[t][n] = res
            else:
                # Compute the discounted future costs recursively
                realized_costs[t][n] = res + lambda_discount *
realized_costs[t + 1][n]
    return realized_costs


def remove_outliers(Buffer: []):
    reduced_buf =  np.array(Buffer, dtype=object)
    buf_to_del = np.array(Buffer, dtype=object)
    reduced_buf = np.ravel([np.sum(np.sum(reduced_buf[:,:,:,1][j]),axis=0)
for j in range( reduced_buf.shape[0])])
    Q1 = np.percentile(reduced_buf, 25, method='midpoint')
    Q3 = np.percentile(reduced_buf, 75,method='midpoint')
    IQR = Q3-Q1
    upper_ar = np.where( reduced_buf >= Q3+1.5*IQR)[0]
    return np.delete(buf_to_del, upper_ar, axis=0)


def remove_outliers_r(Buffer: []):
    reduced_buf = np.zeros(len(Buffer))
    buf_to_del = np.array(Buffer, dtype=object)
    for ib,b in enumerate(Buffer):
        for it,t in enumerate(Buffer[ib]):
            for n in t[1]:
                reduced_buf[ib] += n

    # reduced_buf = np.ravel([np.sum(reduced_buf[j][1]) for j in
range(reduced_buf.shape[0])])
    Q1 = np.percentile(reduced_buf, 25, method='midpoint')
    Q3 = np.percentile(reduced_buf, 75,method='midpoint')
    IQR = Q3-Q1
    upper_ar = np.where( reduced_buf >= Q3+1.5*IQR)[0]
    return np.delete(buf_to_del, upper_ar, axis=0)

def generate_path(init_supply, max_expected_demands_inc, max_supply_inc,
```

```
amount_of_districts, epoch_time_amount):
    n = amount_of_districts
    gen = np.random.default_rng()
    W0 = init_supply, (np.zeros((n,), dtype=int), gen.integers(low=1,
high=100, size=n))

    W = []
    T = epoch_time_amount
    warehouse_items = W0[0]
    for j in range(T):

        realized_demands = gen.integers(W0[1][1]) if len(W) == 0 else
gen.integers(W[-1][1][1])
        warehouse_items += W[-1][0] if len(W) != 0 else 0
        if (warehouse_items < 0):
            realized_demands = np.zeros(n, dtype=int)
            warehouse_items = 0
        k = 0
        while (numpy.sum(realized_demands) > warehouse_items):
            realized_demands = gen.integers(W0[1][1]) if len(W) == 0 else
gen.integers(W[-1][1][1])
            if (k == 100):
                realized_demands = np.zeros(n, dtype=int)
                break
            k = k + 1

        expected_demands = W0[1][1] - realized_demands +
gen.integers(max_expected_demands_inc) if len(W) == 0 else \
            W[-1][1][1] - realized_demands + gen.integers(
                max_expected_demands_inc)  # in real case scenarios obtained
by the officials (volunteers)

        warehouse_items = gen.integers(max_supply_inc) if max_supply_inc>0
else 0 if np.sum(
            realized_demands) == 0 else warehouse_items -
gen.integers(np.sum(realized_demands)) \
            if len(W) == 0 else warehouse_items -
gen.integers(np.sum(realized_demands))
        W.append(( gen.integers(max_supply_inc) if max_supply_inc>0 else 0,
(realized_demands, expected_demands)))
    return W0, W

def define_x_heruistic(S,W, T,n,transport_cost, q ):
    xtnk = np.zeros((T, n, 2), dtype=int)
    Icwx = np.zeros(T, dtype=int)
    Sx = []
    trajectory_buffer = []
    for j in range(T):
        Icws, (It, ht, delt, dt) = S[j]
        # obtain a decision xt
        Icwx[j] = Icws
        for k in range(n):
            # print(S[j][1][2])
            if (S[j][1][2][k] >= np.random.randint(low=1, high=3)):

                xinterim = q[0][1] * np.random.randint(low=1, high=3)

                if (xinterim <= Icwx[j]):
                    xtnk[j][k][0] = xinterim
                    Icwx[j] -= xinterim

        n1 = np.random.randint(low=0, high=n)
```

```
        xtemp = np.minimum(q[1][1], Icwx[j] + xtnk[j][n1][0])

        if (S[j][1][2][n1] >= np.random.randint(low=1, high=3)):
            xtnk[j][n1][0] = 0
            xtnk[j][n1][1] = xtemp
        Sx.append(
            (S[j][0] - np.sum(xtnk[j]), (S[j][1][0] + np.sum(xtnk[j,:],
axis=1), S[j][1][1], S[j][1][2], S[j][1][3])))
        Icwxs, (Itnx, htnx, deltax, dtx) = Sx[-1]
        # arrival of supply, realization of demand, expected demand+--
        sCW, (dh, dt1) = W[j]


        newdel = np.array([l + 1 if It[idx] + np.sum(xtnk[j,:][idx]) <=
dh[idx] else 0 for idx, l in enumerate(
            delt)])  # S[j][1][2][l] + 1 if S[j][1][0][l] +
np.sum(xtnk[j][l]) <= W[j][1][0][l] else 0

        S.append((Icwxs + sCW,
                  (
                      np.maximum(np.zeros(Itnx.size, dtype=int), It +
np.sum(xtnk[j], axis=1) - dh),
                      np.maximum(np.zeros(Itnx.size, dtype=int), dh - It -
np.sum(xtnk[j], axis=1)),
                      newdel,
                      dt1))
                 )
    V = compute_realized_future_costs(Sx, xtnk, transport_cost, q,
lambda_discount=0.9)
    CS = [cost_funct_n(S[j+1], xtnk[j], transport_cost, q) for j in range(T)]
    # print(CS)
    trajectory_buffer.append([(Sx[j], cost_funct_n(S[j], xtnk[j],
transport_cost, q), V[j]) for j in range(T)])
    return S, xtnk, trajectory_buffer

# define_xt_heruistic()

def warmup(max_expected_demands_inc,
           q,
           transport_cost,
            n = 4,
           init_supply=abs(int(np.random.normal(loc=3400, scale=4))),
           max_supply_inc=abs(int(np.random.normal(100, scale=4))),
           buffer_size = 1000,
           T = 6,
           discount_factor=0.9):
    BufferSize = buffer_size
    Buffer = []

    for i in range(BufferSize):
        actual_supply = init_supply
        W0, W =
generate_path(actual_supply,max_expected_demands_inc,max_supply_inc,amount_of
_districts=n, epoch_time_amount=T)
        # warehouse, inv, shortage (amount of unsatisfied users after
distribution in moment t), time of deprivation, expected demand
        S = [(W0[0], (np.zeros(n, dtype=int), np.zeros(n, dtype=int),
np.zeros(n, dtype=int), W0[1][1]))]
        S, xtnk, trajectory_buffer = define_x_heruistic(S, W, T, n,
transport_cost, q)
        Buffer.append(trajectory_buffer)
    Buffer = remove_outliers(Buffer)
    # remove outliers
```

```
    return Buffer


# SGD regressor



# learning process



# Press the green button in the gutter to run the script.
def algo(amount_of_distr = 4,
         initial_warehouse_supply = 2000,
         q = np.array([
        [np.random.randint(low=1, high=4),  # cost per km or speed
         np.random.randint(low=40, high=400)  # capacity
         ],
        [
            np.random.randint(low=1, high=4),
            np.random.randint(low=20, high=200)
        ]
    ]),
    T = 6
    ,max_supply_inc = int(0)
    ,transport_cost = np.random.default_rng().integers(size=4, low=1,
high=20),
    max_expected_demands_inc = [100,100,100,100]
    , buffer_range = 1000):
    print_hi('PyCharm')


    #considering initial supply is provided

    wbuf = warmup(n=amount_of_distr, q=q, transport_cost= transport_cost,
                  init_supply=initial_warehouse_supply,
max_expected_demands_inc=max_expected_demands_inc,
                  max_supply_inc=max_supply_inc,T=T,
buffer_size=buffer_range)

    values_of_dt_for_std = wbuf[:, 0, :, 0]
    std_vals = np.empty((T,amount_of_distr))
    for t in range(T):
        for ni in range(amount_of_distr):
            temp = np.zeros(len(values_of_dt_for_std[:,t]))
            for idx in range(len(values_of_dt_for_std[:,t])):
                temp[idx] = values_of_dt_for_std[:,t][idx][1][3][ni]
            std_vals[t][ni] = np.std(temp)

    features = np.array([[(t[0][1][0], t[0][1][2], np.array(
        [deprivation_cost(k + 1) for k in t[0][1][2]] * np.maximum(0,
t[0][1][3] + 2 * np.std(t[0][1]) - t[0][1][0])))
                         for t in wbuf[j][0]] for j in
range(wbuf.shape[0])])

    y = wbuf[:, 0, :, 1]
      # defined for each n by haversine dist from central warehouse

    weights_coef = compute_weights(features, y)
    # plt.show()

    eps = 0.2
    eps_decay = 0.98
```

```
    alpha = 0.2
    alpha_decay = 0.99
    training_episodes = buffer_range
    divisor = 10
    Br = []
    ValueFunctions = []
    for i in range(training_episodes):
        n  = amount_of_distr
        W0, W =
generate_path(initial_warehouse_supply,max_expected_demands_inc,max_supply_in
c,amount_of_distr, T)
        S = [(W0[0], (np.zeros(n, dtype=int), np.zeros(n, dtype=int),
np.zeros(n, dtype=int), W0[1][1]))]
        # np.array(
        #     [deprivation_cost(k + 1) for k in t[0][1][2]] * np.maximum(0,
t[0][1][3] + 2 * np.std(t[0][1]) - t[0][1][0]))

        tmp_buf = []
        # Parameters (example values; adjust as needed)
        # Parameters (example values; adjust as needed)

        Sx = []
        for j in range(T):
            Icws, (It, ht, delt, dt) = S[j]
            if random.uniform(0,1) < eps:
                calculate_step_by_heruistic_model(Icws, It, S, Sx, W, delt,
dt, ht, j, n, q, tmp_buf, transport_cost)
            else:
                env = Env()
                env.setParam('OutputFlag', 0)
                Icws, (It, ht, delt, dt) = S[j]
                MAX_P = 200000
                # Initialize model
                model = Model("HumanitarianOptimization")
                model.Params.IntFeasTol = 1e-9
                q_gb = np.copy(q)
                # I
                # Decision variables
                K = q_gb.shape[0]
                x = model.addVars(n, K, name="x", vtype=GRB.INTEGER, lb=0)
                I_CW = model.addVar(name="I_CW", vtype=GRB.INTEGER, lb=0)
                I = model.addVars(n, name="I", vtype=GRB.INTEGER, lb=0)
                h = model.addVars(n, name="h", vtype=GRB.CONTINUOUS, lb=0)
                G_x = model.addVars(n, name="G_x", vtype=GRB.CONTINUOUS,
lb=0)
                z_h = model.addVars(n, name="z_h", vtype=GRB.BINARY)


                ceilparam = model.addVars(n, K, vtype=GRB.INTEGER, lb=0)
                model.addConstrs(x[ni, k] >= 0 for ni in range(n) for k in
range(K))
                model.addConstrs(h[ni] >= 0 for ni in range(n))
                model.addConstrs(I[ni] >= 0 for ni in range(n))
                model.addConstrs(G_x[ni] >= 0 for ni in range(n))
                model.addConstr(I_CW >= 0)

                model.addConstr(
                    I_CW == Icws - quicksum(x[ni, k] for ni in range(n) for k
in range(K)),
                    name="InventoryBalanceCW"
                )
                model.addConstrs(
                    (I[ni] == It[ni] + quicksum(x[ni, k] for k in range(K))
```

```
for ni in range(n)),
                    name="InventoryBalanceItem"
                )

                # Auxiliary variable
                model.addConstrs(
                    (G_x[ni] == deprivation_cost(delt[ni] + 1) * h[ni] for ni
in range(n)),
                    name="AuxiliaryGx"
                )
                model.addConstrs(
                    ceilparam[ni, k] >= x[ni, k] / q_gb[k][1] for ni in
range(n) for k in range(K)
                )
                model.addConstrs(
                    ceilparam[ni, k] <= x[ni, k] / q_gb[k][1] + 0.999 for ni
in range(n) for k in range(K)
                )
                # Conditional constraints
                model.addConstrs(
                    (h[ni] >= dt[ni] + 2 * std_vals[j][ni] - I[ni] for ni in
range(n)),
                    name="CondConstraint1"
                )
                model.addConstrs(
                    (h[ni] <= dt[ni] + 2 * std_vals[j][ni] - I[ni] - MAX_P *
(1 - z_h[ni]) for ni in range(n)),
                    name="CondConstraint2"
                )
                model.addConstrs(
                    (h[ni] <= MAX_P * z_h[ni] for ni in range(n)),
                    name="CondConstraint3"
                )
                weighted_t = weights_coef[j]
                model.setObjective(
                    quicksum(deprivation_cost(delt[ni]) * ht[ni] for ni in
range(n)) +
                    quicksum(ceilparam[ni, k]*transport_cost[ni]  for ni in
range(n) for k in range(K))+
                    quicksum(weighted_t[ni][0] * I[ni] + weighted_t[ni][3] +
weighted_t[ni][1]*delt[ni]+weighted_t[ni][2]*G_x[ni] for ni in range(n)),
                    GRB.MINIMIZE
                )

                model.optimize()

                if model.status == GRB.INFEASIBLE:
                    model.feasRelaxS(1, False, False, True)
                    model.optimize()
                #
                if model.status == GRB.OPTIMAL:
                    # print("Optimal solution found!")
                    # for ni in range(n):
                    #     for k in range(K):
                    #         print(f"x[{ni},{k}] = {x[ni, k].X}")
                    #         print(f"cp[{ni},{k}] = {ceilparam[ni, k]}")
                    # print(f"I_CW = {I_CW}")
                    # for ni in range(n):
                    #     print(f"I[{ni}] = {I[ni].X}, h[{ni}] = {h[ni]},
G_x[{ni}] = {G_x[ni]}, z_h[{ni}] = {z_h[ni]}")

                    xnk_opt = model.getAttr('x', x)
                    summing_array = np.empty(n)
```

```
                    xres = np.zeros((n, K))
                    for ni in range(n):
                        for ki in range(K):
                            summing_array[ni] += xnk_opt[ni, ki]
                            xres[ni, ki] = xnk_opt[ni, ki]
                    xsum = np.sum(xres)
                    print("_____________________approx value
function_____________________", quicksum(
                        weighted_t[ni][0] * I[ni].X + weighted_t[ni][3] +
weighted_t[ni][1] * delt[ni] + weighted_t[ni][2] * G_x[ni].X for ni in
                        range(n)))

                    Sx.append((
                        S[j][0] - xsum, (It + np.sum(xres, axis=1), ht, delt,
dt)))
                    Icwxs, (Itnx, htnx, deltax, dtx) = Sx[-1]
                    # arrival of supply, realization of demand, expected
demand+--
                    sCW, (dh, dt1) = W[j]
                    # dh = np.minimum(I+It.X,dh)
                    #as our represantatives adds new data exactly how it is
but now we just simulating
                    # dh+=htnx.as
                    newdel = np.array([l + 1 if It[idx] + np.sum(xres[idx])
<= dh[idx] else 0 for idx, l in enumerate(
                        delt)])  # S[j][1][2][l] + 1 if S[j][1][0][l] +
np.sum(xtnk[j][l]) <= W[j][1][0][l] else 0
                    dh += htnx.astype(dtype=int)
                    dt1+=-It.astype(dtype=int)+dt.astype(dtype=int)+
ht.astype(dtype=int)
                    S.append((Icwxs + sCW,
                              (
                                  np.maximum(np.zeros(Itnx.size, dtype=int),
It + np.sum(xres, axis=1) - dh),
                                  np.maximum(np.zeros(Itnx.size, dtype=int),
dh - It - np.sum(xres, axis=1)),
                                  newdel,
                                  dt1))
                             )
                    tmp_buf.append((Sx[-1], cost_funct_n(S[-1], xres,
transport_cost, q)))
                    model.close()
                    env.close()
                else:
                    calculate_step_by_heruistic_model(Icws, It, S, Sx, W,
delt, dt, ht, j, n, q, tmp_buf, transport_cost)
        Br.append(tmp_buf)
        if(i%divisor==0):
            remove_outliers_r(Br)
            values =
compute_realized_future_costs_using_cost(cost=[tmp_buf[idxj][1] for idxj in
range(len(tmp_buf))], lambda_discount=0.9)

            features1 = np.array([[(ti[0][1][0], ti[0][1][2], np.array(
                [deprivation_cost(k + 1) for k in ti[0][1][2]] *
np.maximum(0,

ti[0][1][3] + 2 * np.std(ti[0][1]) - ti[0][1][

0])))
                                  for ti in Br[j]] for j in range(len(Br))])
            # std_vals = [np.std( values_of_dt_for_std[:,1], axis=1)]
            yt = np.empty((len(Br),len(Br[0]), len(Br[0][0][1])))
```

```
            for r in range(len(Br)):
                for r1 in range(len(Br[r])):
                    for r2 in range(len(Br[r][r1][1])):
                        yt[r][r1][r2] = Br[r][r1][1][r2]

            temp_weights = compute_weights(features1, yt)
            weights_coef = (1-alpha)*weights_coef + alpha*temp_weights
            eps*=eps_decay
            alpha*=alpha_decay
    return weights_coef, std_vals
    # print(feature1, feature2, feature3, y)



def calculate_step_by_heruistic_model(Icws, It, S, Sx, W, delt, dt, ht, j, n,
q, tmp_buf, transport_cost):
    xnk = np.zeros((n, q.shape[0]))
    # obtain a decision xt
    Icwx = Icws
    for k in range(n):
        if (delt[k] >= np.random.randint(low=1, high=3)):
            xinterim = q[0][1] * np.random.randint(low=1, high=3)
            if (xinterim <= Icwx):
                xnk[k][0] = xinterim
                Icwx -= xinterim
    n1 = np.random.randint(low=0, high=n)
    xtemp = min(q[1][1], Icwx + xnk[n1][0])
    if (delt[n1] >= np.random.randint(low=1, high=3)):
        xnk[n1][0] = 0
        xnk[n1][1] = xtemp
    Sx.append(
        (S[j][0] - np.sum(xnk), (It + np.sum(xnk, axis=1), ht, delt, dt)))
    Icwxs, (Itnx, htnx, deltax, dtx) = Sx[-1]
    # arrival of supply, realization of demand, expected demand+--
    sCW, (dh, dt1) = W[j]
    newdel = np.array([l + 1 if It[idx] + np.sum(xnk) <= dh[idx] else 0 for
idx, l in enumerate(
        delt)])  # S[j][1][2][l] + 1 if S[j][1][0][l] + np.sum(xtnk[j][l]) <=
W[j][1][0][l] else 0
    S.append((Icwxs + sCW,
              (
                  np.maximum(np.zeros(Itnx.size, dtype=int), It + np.sum(xnk,
axis=1) - dh),
                  np.maximum(np.zeros(Itnx.size, dtype=int), dh - It +
np.sum(xnk, axis=1)),
                  newdel,
                  dt1))
             )
    tmp_buf.append((Sx[-1], cost_funct_n(S[-1], xnk, transport_cost, q)))
    return xnk


def compute_weights(features, y):
    reg = [[sklearn.linear_model.LinearRegression().fit(features[:, t, :, n],
[yl[n] for yl in y[:, t]]) for n in
            range(len(y[:, t][0]))] for t in range(y.shape[1])]
    weights_coef = np.zeros((len(reg), len(reg[0]), 4))
    for tidx, t in enumerate(reg):
        # reg1.score(features, y)
        for nidx, regr in enumerate(reg[tidx]):
            intercept = regr.intercept_
            for l in range(len( regr.coef_)):
                weights_coef[tidx][nidx][l] = regr.coef_[l]
```

```
            weights_coef[tidx][nidx][-1] = intercept
    return weights_coef


Db2Working


import os
from contextlib import closing

import bcrypt
import psycopg2
import psycopg2.extras

#

#ibmdb2
# db_handler = db2Working.BaseHandler(config.get("db", "database_ip"),
#                                     config.get("db", "database_port"),
#                                     config.get("db", "database_db2_name"),
#                                     config.get("db", "database_db2_login"),
#                                     config.get("db",
"database_db2_password"))

from dotenv import load_dotenv

from datetime import date


class BaseHandler(object):
    #ibm db2
    # def __init__(self, host,port,name , username, passw):
    #     self.conn = ibm_db_dbi.connect('DATABASE={};'
    #                   'HOSTNAME={};'  # 127.0.0.1 or localhost works if it's
local
    #                   'PORT={};'
    #                   'PROTOCOL=TCPIP;'
    #                   'UID={};'
    #                   'PWD={};'.format(name,host,port,username,passw), '',
'')


    #postgre
    def __init__(self):
        load_dotenv()
        #local
        POSTGRES_HOST = os.getenv("POSTGRES_HOST", "localhost")
        POSTGRES_PORT = os.getenv("POSTGRES_PORT", "5432")
        POSTGRES_USER = os.getenv("POSTGRES_USER", "postgres")
        POSTGRES_PASSWORD = os.getenv("POSTGRES_PASSWORD", "password")
        POSTGRES_DB = os.getenv("POSTGRES_DB", "volunteer_db")

        self.conn = psycopg2.connect(
        host=POSTGRES_HOST,
        # port=POSTGRES_PORT,
        user=POSTGRES_USER,
        password=POSTGRES_PASSWORD,
        database=POSTGRES_DB,
        )
 #       remote
#        self.conn = psycopg2.connect(os.environ.get('DATABASE_URL'),
sslmode='require')

    async def verify_user_by_password(self, username, password):
```

```
        #sql = 'select * from location_info';
        with self.conn as conn,
conn.cursor(cursor_factory=psycopg2.extras.DictCursor) as cursor:
            passw = bcrypt.hashpw(password, bcrypt.gensalt(rounds=10,
prefix=b"2a"))
            cursor.execute("SELECT j.*, ad FROM jwt_user_details j "
                           "inner join assigned_district ad on
ad.volunteer_id = j.id"
                           "inner join district d on d.id = ad.district_id"
                           " WHERE j.username = %s AND j.password = %s AND
'VOLUNTEER' in "
                           "(select a.authority from user_authorities ua
inner join authority a on a.id = ua.authority_id where ua.user_id = j.id)",
                           (username, passw))
            user = cursor.fetchone()
        return user

    async def getWarehouseSupplyAmount(self, city):
        with self.conn as conn,
conn.cursor(cursor_factory=psycopg2.extras.DictCursor) as cursor:
            print("City", city)
            cursor.execute("SELECT wl.* from warehouse_location wl where
city_name = %s and time_of_resetting = (select max(w1.time_of_resetting)
from warehouse_location w1 where w1.city_name = wl.city_name)", (city,))
            warehouse_id, capacity, latitude, longitude, _, _ =
cursor.fetchone()
        return warehouse_id, capacity, latitude, longitude

    async def getLocationsWithAsignedWarehouse(self, warehouse):
        with self.conn as conn,
conn.cursor(cursor_factory=psycopg2.extras.DictCursor) as cursor:
            cursor.execute("SELECT * from district where warehouse_id = %s
and active = true", (int(warehouse),))
            vals = cursor.fetchall()
        return vals


    def __del__(self):
        if hasattr(self,'conn'):
            self.conn.close()
```

Main.py

```
import json
import math
import os
import time

from algo import *
from dbwork.dbInit import db_handler
import asyncio
from aiokafka import AIOKafkaConsumer
from datetime import datetime, timedelta
from collections import defaultdict



KAFKA_BROKER = os.getenv("KAFKA_BROKER", "localhost:29092")
KAFKA_TOPIC = "volunteer_reports"
GROUP_ID = "district-group"
```

```
async def process_messages(messages):
    """
    Process the parsed messages grouped by district.
    """
    print(f"Processing messages at {datetime.now()}:")
    for district, reports in messages.items():
        latest_report = reports[-1]  # Assuming messages are ordered; take
the latest
        print(f"District: {district}")
        print("Latest Report:")
        print(json.dumps(latest_report, indent=4))
    print("-" * 50)

async def consume_messages():
    consumer = AIOKafkaConsumer(
        KAFKA_TOPIC,
        bootstrap_servers=KAFKA_BROKER,
        group_id=GROUP_ID,
        value_deserializer=lambda v: json.loads(v.decode("utf-8")),  #
Deserialize JSON
    )

    # Initialize Kafka consumer
    await consumer.start()
    print("Kafka Consumer started. Listening for messages...")
    try:
        district_reports = defaultdict(list)
        # next_run_time = datetime.now() + timedelta(hours=1)

        async for message in consumer:
            # Parse the message
            report = message.value
            try:
                # Ensure the message contains expected fields
                district = report.get("distribution_point", "unknown")
                if all(key in report for key in ["volunteer_id",
"distribution_point", "period", "realized_demands", "expected_demands"]):
                    district_reports[district].append(report)
                else:
                    print(f"Invalid report format: {report}")
            except Exception as e:
                print(f"Failed to parse message: {message.value}. Error:
{e}")
            district_reports.clear()  # Clear stored messages after
processing
            # Check if it's time to process the messages
    finally:
        await consumer.stop()
        return district_reports
    # print("Kafka Consumer stopped.")
        # if datetime.now() >= next_run_time:
        # next_run_time = datetime.now() + timedelta(hours=1)



# Main entry point for running the consumer

if __name__ == '__main__':
    # asyncio.run(consume_messages())
    asyncio.set_event_loop(loop=asyncio.new_event_loop())
    loop = asyncio.get_event_loop()
    warehouse_id, capacity, latitude, longitude = loop.run_until_complete(
db_handler.getWarehouseSupplyAmount("Sumy"))
```

```
    vals =
loop.run_until_complete(db_handler.getLocationsWithAsignedWarehouse(warehouse
_id))

    amount_of_distr = len(vals)
    initial_warehouse_supply = capacity
    q = np.array([
        [np.random.randint(low=1, high=4),  # cost per km or speed
         np.random.randint(low=40, high=400)  # capacity
         ],
        [
            1,
            np.random.randint(low=20, high=200)
        ]
    ])
    #rad of earth, km
    R = 6371
    transport_cost = np.empty(amount_of_distr)
    for i in range(len(vals)):
        dlon = math.radians( vals[i][3]-longitude)
        dlat = math.radians( vals[i][2]-latitude)

        lat1 = math.radians(latitude)
        lat2 = math.radians(vals[i][2])
        a =
math.sin(dlat/2)*math.sin(dlat/2)+math.sin(dlon/2)*math.sin(dlon/2)*math.cos(
lat2)
        c = 2*math.asin(math.sqrt(a))
        transport_cost[i] = R*c

    districts_d = {}
    for i in range(len(vals)):
        districts_d[vals[i][0]] = i

    initial_demand_of_monitoring = [i[4] for i in vals]
    T = 8
    max_expected_demands_inc = [int(float(i) / (T * random.uniform(1, 3)))
for i in initial_demand_of_monitoring]
    max_supply_inc = int(0)
    Sx = []
    # print(algo())
    weights, std_vals = algo(amount_of_distr = amount_of_distr,
                             initial_warehouse_supply =
initial_warehouse_supply,
                             q = q, T = T,
                             max_supply_inc = 0,
                             transport_cost = transport_cost,
                             max_expected_demands_inc =
max_expected_demands_inc,
                             buffer_range=1000)
    env = Env()
    env.setParam('OutputFlag', 0)

    history = {}
    n = amount_of_distr
    #time of sleep eg 1 hour
    sleeping_time = 1000*60*60
    W0, W = generate_path(initial_warehouse_supply, max_expected_demands_inc,
max_supply_inc, amount_of_distr, T)
    S = [(W0[0], (np.zeros(amount_of_distr, dtype=int),
np.zeros(amount_of_distr, dtype=int), np.zeros(n, dtype=int), W0[1][1]))]
    for j in range(T):
        # timeout wait until....
```

```
        print("Sleeping on ", j)
        Icws, (It, ht, delt, dt) = S[j]

        MAX_P = 200000
        # Initialize model
        model = Model("HumanitarianOptimization")
        model.Params.IntFeasTol = 1e-9
        q_gb = np.copy(q)
        # I
        # Decision variables
        K = q_gb.shape[0]
        x = model.addVars(n, K, name="x", vtype=GRB.INTEGER, lb=0)
        I_CW = model.addVar(name="I_CW", vtype=GRB.INTEGER, lb=0)
        I = model.addVars(n, name="I", vtype=GRB.INTEGER, lb=0)
        h = model.addVars(n, name="h", vtype=GRB.CONTINUOUS, lb=0)
        G_x = model.addVars(n, name="G_x", vtype=GRB.CONTINUOUS, lb=0)
        z_h = model.addVars(n, name="z_h", vtype=GRB.BINARY)

        ceilparam = model.addVars(n, K, vtype=GRB.INTEGER, lb=0)
        model.addConstrs(x[ni, k] >= 0 for ni in range(n) for k in range(K))
        model.addConstrs(h[ni] >= 0 for ni in range(n))
        model.addConstrs(I[ni] >= 0 for ni in range(n))
        model.addConstrs(G_x[ni] >= 0 for ni in range(n))
        model.addConstr(I_CW >= 0)

        model.addConstr(
            I_CW == Icws - quicksum(x[ni, k] for ni in range(n) for k in
range(K)),
            name="InventoryBalanceCW"
        )
        model.addConstrs(
            (I[ni] == It[ni] + quicksum(x[ni, k] for k in range(K)) for ni in
range(n)),
            name="InventoryBalanceItem"
        )

        # Auxiliary variable
        model.addConstrs(
            (G_x[ni] == deprivation_cost(delt[ni] + 1) * h[ni] for ni in
range(n)),
            name="AuxiliaryGx"
        )
        model.addConstrs(
            ceilparam[ni, k] >= x[ni, k] / q_gb[k][1] for ni in range(n) for
k in range(K)
        )
        model.addConstrs(
            ceilparam[ni, k] <= x[ni, k] / q_gb[k][1] + 0.999 for ni in
range(n) for k in range(K)
        )
        # Conditional constraints
        model.addConstrs(
            (h[ni] >= dt[ni] + 2 * std_vals[j][ni] - I[ni] for ni in
range(n)),
            name="CondConstraint1"
        )
        model.addConstrs(
            (h[ni] <= dt[ni] + 2 * std_vals[j][ni] - I[ni] - MAX_P * (1 -
z_h[ni]) for ni in range(n)),
            name="CondConstraint2"
        )
        model.addConstrs(
            (h[ni] <= MAX_P * z_h[ni] for ni in range(n)),
```

```
            name="CondConstraint3"
        )
        weighted_t = weights[j]
        model.setObjective(
            quicksum(deprivation_cost(delt[ni]) * ht[ni] for ni in range(n))
+
            quicksum(ceilparam[ni, k] * transport_cost[ni] for ni in range(n)
for k in range(K)) +
            quicksum(
                weighted_t[ni][0] * I[ni] + weighted_t[ni][3] +
weighted_t[ni][1] * delt[ni] + weighted_t[ni][2] * G_x[ni]
                for ni in range(n)),
            GRB.MINIMIZE
        )

        model.optimize()

        if model.status == GRB.INFEASIBLE:
            model.feasRelaxS(1, False, False, True)
            model.optimize()
        #
        if model.status == GRB.OPTIMAL:
            print("Optimal solution found!")
            for ni in range(n):
                for k in range(K):
                    print(f"x[{ni},{k}] = {x[ni, k].X}")
                    print(f"cp[{ni},{k}] = {ceilparam[ni, k]}")
            print(f"I_CW = {I_CW}")
            for ni in range(n):
                print(f"I[{ni}] = {I[ni].X}, h[{ni}] = {h[ni]}, G_x[{ni}] =
{G_x[ni]}, z_h[{ni}] = {z_h[ni]}")

            xnk_opt = model.getAttr('x', x)
            summing_array = np.empty(n)
            xres = np.zeros((n, K))
            for ni in range(n):
                for ki in range(K):
                    summing_array[ni] += xnk_opt[ni, ki]
                    xres[ni, ki] = xnk_opt[ni, ki]
            xsum = np.sum(xres)
            print("_____________________approx value
function_____________________", quicksum(
                weighted_t[ni][0] * I[ni].X + weighted_t[ni][3] +
weighted_t[ni][1] * delt[ni] + weighted_t[ni][2] * G_x[
                    ni].X for ni in
                range(n)))

            Sx.append((
                S[j][0] - xsum, (It + np.sum(xres, axis=1), ht, delt, dt)))
            Icwxs, (Itnx, htnx, deltax, dtx) = Sx[-1]
            # arrival of supply, realization of demand, expected demand+--

            sCW, (dh, dt1) = W[j]
            dh+=ht.astype(dtype=int) # for testing purposes
            newdel = np.array([l + 1 if It[idx] + np.sum(xres[idx]) <=
dh[idx] else 0 for idx, l in enumerate(
                delt)])  # S[j][1][2][l] + 1 if S[j][1][0][l] +
np.sum(xtnk[j][l]) <= W[j][1][0][l] else 0
            S.append((Icwxs + sCW,
                      (
                          np.maximum(np.zeros(Itnx.size, dtype=int), It +
np.sum(xres, axis=1) - dh),
                          np.maximum(np.zeros(Itnx.size, dtype=int), dh - It
```

```
- np.sum(xres, axis=1)),
                            newdel,
                            dt1))
                    )
            xnkI = xres
            model.close()
            env.close()
            print(xnkI)
            history["time "+str(j)] = xnkI, Itnx, htnx, Icws
        else:
            xnkI = calculate_step_by_heruistic_model(Icws, It, S, Sx, W,
delt, dt, ht, j, n, q, [], transport_cost)
            print(xnkI)
            history["time " + str(j)] = xnkI
        # time.sleep(sleeping_time)
    print(history)


import json
import math
import os
import time

from algo import *
from dbwork.dbInit import db_handler
import asyncio
from aiokafka import AIOKafkaConsumer
from datetime import datetime, timedelta
from collections import defaultdict



KAFKA_BROKER = os.getenv("KAFKA_BROKER", "localhost:29092")
KAFKA_TOPIC = "volunteer_reports"
GROUP_ID = "district-group"

async def process_messages(messages):
    """
    Process the parsed messages grouped by district.
    """
    print(f"Processing messages at {datetime.now()}:")
    for district, reports in messages.items():
        latest_report = reports[-1]  # Assuming messages are ordered; take
the latest
        print(f"District: {district}")
        print("Latest Report:")
        print(json.dumps(latest_report, indent=4))
    print("-" * 50)

async def consume_messages():
    consumer = AIOKafkaConsumer(
        KAFKA_TOPIC,
        bootstrap_servers=KAFKA_BROKER,
        group_id=GROUP_ID,
        value_deserializer=lambda v: json.loads(v.decode("utf-8")),  #
Deserialize JSON
    )

    # Initialize Kafka consumer
    await consumer.start()
    print("Kafka Consumer started. Listening for messages...")
    try:
        district_reports = defaultdict(list)
```

```
        # next_run_time = datetime.now() + timedelta(hours=1)

        async for message in consumer:
            # Parse the message
            report = message.value
            try:
                # Ensure the message contains expected fields
                district = report.get("distribution_point", "unknown")
                if all(key in report for key in ["volunteer_id",
"distribution_point", "period", "realized_demands", "expected_demands"]):
                    district_reports[district].append(report)
                else:
                    print(f"Invalid report format: {report}")
            except Exception as e:
                print(f"Failed to parse message: {message.value}. Error:
{e}")
            district_reports.clear()  # Clear stored messages after
processing
            # Check if it's time to process the messages
    finally:
        await consumer.stop()
        return district_reports
    # print("Kafka Consumer stopped.")
        # if datetime.now() >= next_run_time:
        # next_run_time = datetime.now() + timedelta(hours=1)



# Main entry point for running the consumer

if __name__ == '__main__':
    # asyncio.run(consume_messages())
    asyncio.set_event_loop(loop=asyncio.new_event_loop())
    loop = asyncio.get_event_loop()
    warehouse_id, capacity, latitude, longitude = loop.run_until_complete(
db_handler.getWarehouseSupplyAmount("Sumy"))
    vals =
loop.run_until_complete(db_handler.getLocationsWithAsignedWarehouse(warehouse
_id))

    amount_of_distr = len(vals)
    initial_warehouse_supply = capacity
    q = np.array([
        [np.random.randint(low=1, high=4),  # cost per km or speed
         np.random.randint(low=40, high=400)  # capacity
         ],
        [
            1,
            np.random.randint(low=20, high=200)
        ]
    ])
    #rad of earth, km
    R = 6371
    transport_cost = np.empty(amount_of_distr)
    for i in range(len(vals)):
        dlon = math.radians( vals[i][3]-longitude)
        dlat = math.radians( vals[i][2]-latitude)

        lat1 = math.radians(latitude)
        lat2 = math.radians(vals[i][2])
        a =
math.sin(dlat/2)*math.sin(dlat/2)+math.sin(dlon/2)*math.sin(dlon/2)*math.cos(
lat2)
```

```
        c = 2*math.asin(math.sqrt(a))
        transport_cost[i] = R*c

    # transport
    # initial_demand_of_monitoring = [max(1, math.ceil(
    #     random.normalvariate(initial_warehouse_supply,
initial_warehouse_supply / 100) / (
    #         random.uniform(1, 2)))) * math.ceil(random.uniform(1, 3)) for i
in range(amount_of_distr)]
    districts_d = {}
    for i in range(len(vals)):
        districts_d[vals[i][0]] = i

    initial_demand_of_monitoring = [i[4] for i in vals]
    T = 8
    max_expected_demands_inc = [int(float(i) / (T * random.uniform(1, 3)))
for i in initial_demand_of_monitoring]
    max_supply_inc = int(0)
    Sx = []
    # print(algo())
    weights, std_vals = algo(amount_of_distr = amount_of_distr,
                             initial_warehouse_supply =
initial_warehouse_supply,
                             q = q, T = T,
                             max_supply_inc = 0,
                             transport_cost = transport_cost,
                             max_expected_demands_inc =
max_expected_demands_inc,
                             buffer_range=1000)
    print(weights)
    env = Env()
    env.setParam('OutputFlag', 0)
    #TODO
    # basic approach, should be improved by using time intervals
    history = {}
    n = amount_of_distr
    #time of sleep eg 1 hour
    sleeping_time = 1000*60*60
    W0, W = generate_path(initial_warehouse_supply, max_expected_demands_inc,
max_supply_inc, amount_of_distr, T)
    S = [(W0[0], (np.zeros(amount_of_distr, dtype=int),
np.zeros(amount_of_distr, dtype=int), np.zeros(n, dtype=int), W0[1][1]))]
    for j in range(T):
        # timeout wait until....
        print("Sleeping on ", j)
        Icws, (It, ht, delt, dt) = S[j]

        MAX_P = 200000
        # Initialize model
        model = Model("HumanitarianOptimization")
        model.Params.IntFeasTol = 1e-9
        q_gb = np.copy(q)
        # I
        # Decision variables
        K = q_gb.shape[0]
        x = model.addVars(n, K, name="x", vtype=GRB.INTEGER, lb=0)
        I_CW = model.addVar(name="I_CW", vtype=GRB.INTEGER, lb=0)
        I = model.addVars(n, name="I", vtype=GRB.INTEGER, lb=0)
        h = model.addVars(n, name="h", vtype=GRB.CONTINUOUS, lb=0)
        G_x = model.addVars(n, name="G_x", vtype=GRB.CONTINUOUS, lb=0)
        z_h = model.addVars(n, name="z_h", vtype=GRB.BINARY)

        ceilparam = model.addVars(n, K, vtype=GRB.INTEGER, lb=0)
```

```
        model.addConstrs(x[ni, k] >= 0 for ni in range(n) for k in range(K))
        model.addConstrs(h[ni] >= 0 for ni in range(n))
        model.addConstrs(I[ni] >= 0 for ni in range(n))
        model.addConstrs(G_x[ni] >= 0 for ni in range(n))
        model.addConstr(I_CW >= 0)

        model.addConstr(
            I_CW == Icws - quicksum(x[ni, k] for ni in range(n) for k in
range(K)),
            name="InventoryBalanceCW"
        )
        model.addConstrs(
            (I[ni] == It[ni] + quicksum(x[ni, k] for k in range(K)) for ni in
range(n)),
            name="InventoryBalanceItem"
        )

        # Auxiliary variable
        model.addConstrs(
            (G_x[ni] == deprivation_cost(delt[ni] + 1) * h[ni] for ni in
range(n)),
            name="AuxiliaryGx"
        )
        model.addConstrs(
            ceilparam[ni, k] >= x[ni, k] / q_gb[k][1] for ni in range(n) for
k in range(K)
        )
        model.addConstrs(
            ceilparam[ni, k] <= x[ni, k] / q_gb[k][1] + 0.999 for ni in
range(n) for k in range(K)
        )
        # Conditional constraints
        model.addConstrs(
            (h[ni] >= dt[ni] + 2 * std_vals[j][ni] - I[ni] for ni in
range(n)),
            name="CondConstraint1"
        )
        model.addConstrs(
            (h[ni] <= dt[ni] + 2 * std_vals[j][ni] - I[ni] - MAX_P * (1 -
z_h[ni]) for ni in range(n)),
            name="CondConstraint2"
        )
        model.addConstrs(
            (h[ni] <= MAX_P * z_h[ni] for ni in range(n)),
            name="CondConstraint3"
        )
        weighted_t = weights[j]
        model.setObjective(
            quicksum(deprivation_cost(delt[ni]) * ht[ni] for ni in range(n))
+
            quicksum(ceilparam[ni, k] * transport_cost[ni] for ni in range(n)
for k in range(K)) +
            quicksum(
                weighted_t[ni][0] * I[ni] + weighted_t[ni][3] +
weighted_t[ni][1] * delt[ni] + weighted_t[ni][2] * G_x[ni]
                for ni in range(n)),
            GRB.MINIMIZE
        )

        model.optimize()

        if model.status == GRB.INFEASIBLE:
            model.feasRelaxS(1, False, False, True)
```

```
            model.optimize()
        #
        if model.status == GRB.OPTIMAL:
            print("Optimal solution found!")
            for ni in range(n):
                for k in range(K):
                    print(f"x[{ni},{k}] = {x[ni, k].X}")
                    print(f"cp[{ni},{k}] = {ceilparam[ni, k]}")
            print(f"I_CW = {I_CW}")
            for ni in range(n):
                print(f"I[{ni}] = {I[ni].X}, h[{ni}] = {h[ni]}, G_x[{ni}] =
{G_x[ni]}, z_h[{ni}] = {z_h[ni]}")

            xnk_opt = model.getAttr('x', x)
            summing_array = np.empty(n)
            xres = np.zeros((n, K))
            for ni in range(n):
                for ki in range(K):
                    summing_array[ni] += xnk_opt[ni, ki]
                    xres[ni, ki] = xnk_opt[ni, ki]
            xsum = np.sum(xres)

            Sx.append((
                S[j][0] - xsum, (It + np.sum(xres, axis=1), ht, delt, dt)))
            Icwxs, (Itnx, htnx, deltax, dtx) = Sx[-1]
            # arrival of supply, realization of demand, expected demand+--
            # msg =
            # dh = msg[]
            sCW, (dh, dt1) = W[j]
            dh+=ht.astype(dtype=int) # for testing purposes

            #KAFKA APPROACH


            # dh = np.empty(amount_of_distr)
            # dt1 = np.empty(amount_of_distr)
            # sCW, (_, _) = W[j]
            # messages = loop.run_until_complete( consume_messages())
            # for district, reports in messages.items():
            #     latest_report = reports[-1]  # Assuming messages are
ordered; take the latest
            #     dh[districts_d[district]] =
latest_report["realized_demands"]
            #     dt[districts_d[district]] =
latest_report["expected_demands"]
            # print("-" * 50)

            newdel = np.array([l + 1 if It[idx] + np.sum(xres[idx]) <=
dh[idx] else 0 for idx, l in enumerate(
                delt)])  # S[j][1][2][l] + 1 if S[j][1][0][l] +
np.sum(xtnk[j][l]) <= W[j][1][0][l] else 0
            # dh += htnx.astype(dtype=int)
            # dt1 += -It.astype(dtype=int) + dt.astype(dtype=int) +
ht.astype(dtype=int)
            S.append((Icwxs + sCW,
                      (
                          np.maximum(np.zeros(Itnx.size, dtype=int), It +
np.sum(xres, axis=1) - dh),
                          np.maximum(np.zeros(Itnx.size, dtype=int), dh - It
- np.sum(xres, axis=1)),
                          newdel,
                          dt1))
                     )
```

```
            xnkI = xres
            model.close()
            env.close()
            print(xnkI)
            history["time "+str(j)] = xnkI, Itnx, htnx, Icws
        else:
            xnkI = calculate_step_by_heruistic_model(Icws, It, S, Sx, W,
delt, dt, ht, j, n, q, [], transport_cost)
            print(xnkI)
            history["time " + str(j)] = xnkI
        # time.sleep(sleeping_time)
    print(history)
```