НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
«___»_____2025 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»
спеціальності 121 Інженерія програмного забезпечення

на тему: «Генерування часових рядів для імітації навантаження в системах мікросервісів»

Виконав:
студент IV курсу, групи ТВ-13
Поддубій Ярослав Васильович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник:
асистент, PhD, Дмитренко О. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:
професор, д.т.н., Скулиш М. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
«___»_____2025р.

ЗАВДАННЯ

на дипломну роботу студенту

Поддубію Ярославу Васильовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: Генерування часових рядів для імітації навантаження в системах мікросервісів
керівник роботи Дмитренко Олександра Анатоліївна, асистент, PhD
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «02» __червня__ 2025р. №_1875-с_____
2. Строк подання студентом роботи « 09 » червня 2025р.
3. Вихідні дані до роботи: мова програмування Python, середовище розробки PyCharm, QtDesigner, база даних SQLite

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити): розглянути методи генерування часових рядів, обрати модель для генерування, розробити алгоритм генерування часових рядів, що імітують використання серверних ресурсів мікросервісами
5. Перелік ілюстративного матеріалу:Use Case, DFD, IDEF0 діаграми, IDE PyCharm, QtDesigner, інтерфейс користувача, матриці переходів, приклади файлів, графіки, алгоритми, модель БД
6. Дата видачі завдання «30» жовтня 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	30.10.2024	Виконано
2	Дослідження предметної області	31.10.2024 – 01.11.2024	Виконано
3	Дослідження існуючих рішень	02.11.2024 – 01.12.2024	Виконано
4	Постановка вимог до проєктування системи	02.12.2024 – 12.01.2025	Виконано
5	Розробка програмного продукту	13.01.2025 – 11.05.2025	Виконано
6	Тестування	12.05.2025 – 15.05.2025	Виконано
7	Захист програмного продукту	12.05.2025 – 15.05.2025	Виконано
8	Оформлення дипломної роботи	19.05.2025 – 01.06.2025	Виконано
9	Передзахист	02.06.2025 – 06.06.2025	Виконано
10	Захист	16.06.2025 – 27.06.2025	

Студент _____ Ярослав Поддубій
(підпис) (ім’я, прізвище)

Керівник роботи _____ Олександра Дмитренко
(підпис) (ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка містить 51 сторінку, 53 рисунки, 1 таблицю, 2 додатки та 18 посилань.

Метою роботи є розробка інструменту для генерації часових рядів вжитку серверних ресурсів програмними продуктами, зокрема оперативної пам’яті, процесорного та канального ресурсу.

Практичне значення одержаних результатів полягає в генерації часових рядів, які імітують навантаження в системах мікросервісів, що дозволить тестувати алгоритми автоматичного розподілу ресурсів залежно від його моделі навантаження.

Ключові слова: часові ряди, хмарні технології, мікросервіси, мікросервісна архітектура, навантаження, ланцюги Маркова.

ABSTRACT

The explanatory note contains 51 pages, 53 figures, 1 table, 2 appendices and 18 references.

The purpose of the work is to develop a tool for generating time series of server resource consumption by software products, in particular RAM, processor and channel resources.

The practical significance of the results obtained lies in the generation of time series that simulate the load in microservice systems, which will allow testing algorithms for automatic resource allocation depending on its load model.

Keywords: time series, cloud technologies, microservices, microservice architecture, load, Markov chains.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД	База даних
IDE	Інтегроване середовище розробки
GUI	Графічний інтерфейс користувача
ОЗП	Оперативний запам’ятовуючий пристрій
ПЗ	Програмне забезпечення
ПП	Програмний продукт

ВСТУП

У наш час програмне забезпечення є невід'ємною частиною нашого життя, яка дозволяє полегшувати виконання різних справ. ПЗ застосовується абсолютно всюди: від комерційних платформ та додатків до сервісів, які існують для того, щоб розважати людей та допомагати їм проводити час весело.

Доволі розповсюдженою архітектурою програмних засобів є мікросервісна архітектура. Даний тип архітектури програмного забезпечення реалізує розподілену систему, або систему поділену на функціональні частини, тобто її принцип полягає в побудові ПЗ на основі невеликих сервісів, кожен з яких виконує певну бізнес-функцію та є незалежним від інших сервісів системи[1, 2]. Ці сервіси комунікують між собою за допомогою різних механізмів, таких як, наприклад, HTTP та gRPC[2]. Завдяки розбиттю продукту на таку сукупність незалежних сервісів, мікросервісні системи дозволяють розгортати певні сервіси для вирішення поставлених задач не впливаючи на інші компоненти системи. Ця властивість також дозволяє масштабувати лише ті процеси системи, які піддаються найбільшому навантаженню, а не всю систему загалом.

Для того, щоб протестувати роботу систем мікросервісів, потрібно мати певний набір даних про навантаження сервісів. Тому ідеєю дипломної роботи стала розробка алгоритму для генерування часових рядів, які імітують навантаження в системах мікропослуг.

1 ПОСТАНОВКА ЗАДАЧІ

Розвиток хмарних технологій відкрив нову еру у створенні програмного забезпечення на початку 2000-х років. Справжній прорив у сфері розробки й тестування хмарних застосунків відбувся у 2013 році з появою Docker. Впровадження контейнеризації, а згодом і системи оркестрації Kubernetes, стало потужним поштовхом для поширення мікросервісної архітектури. Це дало змогу ефективно і гнучко моделювати, розгортати та масштабувати мікросервіси в хмарному середовищі[3].

Хмарна інфраструктура являє собою сукупність апаратного й програмного забезпечення, що забезпечує роботу хмарних обчислень. Вона включає сервери, системи зберігання даних, мережеве обладнання, програмне забезпечення для віртуалізації, а також інструменти управління та сервіси. Ключову роль відіграє віртуалізація, яка дозволяє створювати віртуальні ресурси (наприклад, віртуальні машини) поверх фізичного обладнання. Ці ресурси абстрагуються і надаються користувачам через інтерфейси API, командний рядок або графічні панелі[5].

Набір даних, що містить інформацію про навантаження певного сервісу в системі мікросервісів може бути використаний для багатьох цілей. Зокрема, ці дані є корисними для тестування алгоритму групування мікросервісів.

Задача групування мікросервісів полягає в формуванні таких груп мікросервісів, які б використовували максимально можливі ресурси серверів.

У мікросервісних системах часто спостерігається нерівномірне навантаження на сервіси. Одні використовуються частіше, інші рідше, певні мікросервівси можуть вимагати більших ресурсів серверу для виконання своєї задачі, а інші можуть використовувати зовсім малу їх кількість.

Неправильне групування мікросервісів на серверах може призвести до неправильного розподілення ресурсів, внаслідок чого певні сервери можуть мати великий запас обчислювальних ресурсів, що означає велику кількість невикористаних ресурсів та велику кількість серверів.

Реалізація рішення такої задачі дозволяє оптимально використовувати ресурси серверів, шляхом розміщення на них такого набору мікросервісів, який би використовував їх максимально ефективно, що дозволяє зменшити кількість серверів.

Для тестування роботи такого алгоритму потрібно мати вхідні набори даних, які містять дані про навантаження на сервери різними мікросервісами. Задача даної дипломної роботи полягає саме у реалізації ПЗ, яке дозволятиме генерувати часові ряди, які імітуватимуть використання мікросервісами серверних ресурсів.

Висновки до розділу 1

У першому розділі розглянуто розвиток хмарної інфраструктури та застосування мікросервісної архітектури.

Описано проблему нерівномірного завантаження серверів та спосіб вирішення цієї проблеми – створення алгоритму групування мікросервісів для формування оптимальних груп мікропослуг. Рішення дозволить оптимізувати навантаження на серверні ресурси.

Сформульовано мету дипломної роботи – розробка алгоритму генерування часових рядів для імітації навантаження систем мікросервісів. Такий алгоритм дозволятиме тестувати описані алгоритми групування мікропослуг.

2 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Для реалізації поставленої задачі, потрібно спочатку як слід проаналізувати предметну область. Потрібно розглянути можливі шляхи вирішення задачі, існуючі рішення, скласти список вимог та розробити архітектуру системи.

2.1 Методи генерування часових рядів

Для генерування часових рядів існують вже розроблені методи, що можуть базуватися на обробці статистичних та аналітичних даних, а також на нейронних мережах та машинному навчанні.

2.1.1 ARIMA

ARIMA (AutoRegressive Integrated Moving Average) являє собою метод, який використовується для аналізу часових рядів і передбачення їхніх майбутніх значень. Він поєднує три основні компоненти: авторегресію (AR), ковзне середнє (MA) та інтегрування (I), яке полягає у перетворенні нестаціонарного ряду на стаціонарний за допомогою диференціювання [6, 7].

Такий метод моделює часові ряди на основі попередніх значень у вхідному наборі даних та є дієвим для генерування часових рядів, для яких властива стабільність та лінійність. Це означає, що цей підхід здебільшого застосовується для даних, у яких не спостерігається різких змін значень.

Метод ARIMA базується на підході Бокса-Дженкінса, який складається з чотирьох основних етапів:

1) ідентифікація – перевірка стаціонарності ряду, визначення необхідного порядку диференціювання, аналіз автокореляційної (ACF) та часткової автокореляційної (PACF) функцій для вибору параметрів;

2) оцінювання – підбір параметрів моделі на основі навчальних даних;

3) діагностична перевірка – оцінка якості побудованої моделі та перевірка залишків;

4) прогнозування – використання моделі для передбачення майбутніх значень.

Модель описується трьома параметрами:

1) p – порядок авторегресії (AR);

2) d – порядок диференціювання (I);

3) q – порядок ковзного середнього (MA).

Формально записується як: ARIMA(p, d, q).

2.1.2 Random Forest

Random Forest – це популярний алгоритм машинного навчання, розроблений Лео Брайманом та Адель Катлер, що поєднує результати багатьох дерев рішень для забезпечення більш точної відповіді. Цей алгоритм використовують як для задач класифікації, так і для регресії[8, 9].

Алгоритм заснований на ансамблевому підході, зокрема беггінгу (bootstrap aggregation). Кожне дерево у «лісі» тренується на випадковій підвибірці навчальних даних із поверненням. На кожному вузлі дерева вибирається випадкова підмножина ознак (feature bagging), що зменшує кореляцію між деревами. У результаті модель зменшує ризик перенавчання (overfitting) та варіації, покращуючи загальну точність.

Ключовими параметрами даного алгоритму є:

- кількість дерев у лісі;
- кількість ознак, що розглядаються при розбитті вузла – контроль ступеня випадковості;
- мінімальний розмір вузла – впливає на глибину дерева.

2.1.3 RNN

Рекурентні нейронні мережі (RNN) – це тип нейронних мереж глибокого навчання, розроблених для роботи з даними часових рядів. Вони використовують довгострокову пам’ять, що дозволяє виявляти та моделювати довгострокові залежності в часових рядах[10, 11].

Такий підхід широко використовується для моделювання часових рядів, обробки природньої мови (NLP), розпізнавання мовлення, тощо.

RNN використовують прихований стан (пам’ять), який оновлюється при кожному заданні нових вхідних даних. На кожному кроці вхідні дані та прихований стан використовуються для обчислення нового прихованого стану, а також використовуються однакові вагові коефіцієнти, які з часом оновлюються шляхом зворотного поширення помилки.

2.1.4 Ланцюги Маркова

Ланцюги Маркова реалізують математичну модель, яка описує систему, що переходить із одного стану в інший у межах певного набору станів. Ключовою особливістю ланцюгів Маркова є властивість Маркова, яка полягає в тому, що ймовірність переходу до наступного стану залежить лише від стану, в якому система перебуває в даний момент часу і не залежить від послідовності попередніх подій. Генеруючи велику кількість можливих послідовностей станів, ланцюги Маркова дозволяють стохастично моделювати поведінку систем і наближено оцінювати результати та ризики[12, 13].

Для наочного подання ланцюгів Маркова використовують матриці переходів. Матриця переходів – це квадратна матриця, яка відображає ймовірності переходу системи з одного стану в інший (рис. 2.1).

$$P = \begin{pmatrix} 0.4 & 0.3 & 0.2 & 0.1 \\ 0.2 & 0.5 & 0.2 & 0.1 \\ 0.1 & 0.3 & 0.4 & 0.2 \\ 0.1 & 0.1 & 0.4 & 0.4 \end{pmatrix}$$

Рисунок 2.1 – Матриця переходів

Рядки такої матриці відповідають за поточний стан системи, а стовпчики – за наступний, відповідно на перехресті рядків і стовпчиків відображаються ймовірності переходу системи з поточного стану в наступний.

Ланцюги Маркова підходять для моделювання процесів, де наступний стан не залежить від попередніх, що саме підходить для моделювання навантаження на мікросервіси, адже їх навантаження в майбутньому не залежить від їх навантаження в минулому.

Також, перевагою використання саме ланцюгів Маркова є те, що такий підхід дозволяє генерувати дані без реальних датасетів. Використовуючи цей метод можна вручну налаштовувати кількість станів системи та значення матриці переходів та генерувати часові ряди на основі самостійно налаштованої матриці, а не на основі матриці, яка налаштовується відповідно до вхідного файлу з набором даних.

2.2 Аналіз вимог

Для проектування продукту варто чітко визначити список вимог, висунутих до нього. Для розроблюваного додатку для генерування часових рядів, що імітують навантаження в мікросервісних системах було вирішено реалізувати наступний функціонал:

- вибір типу налаштування матриці переходів (з конфігураційного файлу, або з датасету);

- вибір параметрів для генерування часових рядів (кількість станів навантаження, дата і час початку та кінця генерування, кількість мікросервісів, часовий крок);
- генерування часових рядів за заданими параметрами;
- перегляд графіку згенерованого часового ряду;
- перегляд графіку вибраного датасету;
- збереження часового ряду в БД;
- збереження часового ряду у вигляді .csv файлу;

Для візуалізації функцій, процесів, потоку інформації та загалом схеми роботи системи зручно використовувати UML діаграми.

Універсальна мова моделювання (Unified Modelling Language або UML) – це мова позначень або побудови діаграм, призначена для визначення, візуалізації і документування моделей зорієнтованих на об’єкти систем програмного забезпечення. У конструкціях UML немає чіткого порядку виконання користувачем дій в розроблюваній системі, натомість UML дозволяє наочно показати компонування системи та полегшити подальшу реалізацію функціональних вимог та співпрацю з іншими розробниками, для яких важливо пояснити, як система повинна працювати [14]. Для візуалізації висунутих до системи вимог та її архітектури було створено три діаграми: діаграма варіантів використання (Use Case diagram), діаграма потоків даних (DFD diagram) і IDEF0 діаграма.

2.2.1 Use Case diagram

Діаграма варіантів використання (діаграма прецедентів) – це UML діаграма, яка дозволяє візуально показати функціональні вимоги системи з точки зору користувача (актора)[15]. Діаграми прецедентів розробляються на ранній стадії проєктування системи та призначені для простого пояснення роботи системи та

формування функціональних вимог до системи. Складовими діаграми варіантів використання є такі три об’єкти, як:

1) актор;
2) прецедент (варіант використання, use case);
3) зв’язки.

Актор є сутністю, що взаємодіє з системою, але не належить до неї, тобто актор – це узагальнена група користувачів системи. Актори на діаграмах прецедентів зображуються у вигляді чоловічків (рис. 2.2).

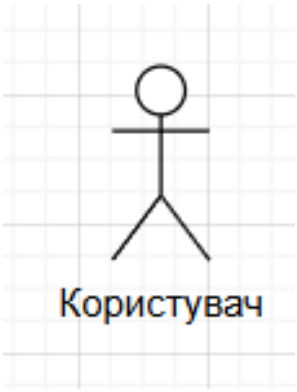


Рисунок 2.2 – Позначення актора на Use Case діаграмі

Прецеденти є діями (функціями), які може здійснювати система та які може використовувати актор. Цей елемент діаграми зображується як еліпс, який містить назву варіанту використання (рис. 2.3).

Рисунок 2.3 – Зображення варіанту використання

Між акторами та прецедентами, а також між самими прецедентами існують зв’язки (рис. 2.4), які бувають трьох наступних типів:

1) асоціація – звичайний зв’язок між актором та прецедентом;

2) розширення (extend) – показує, що одна дія розширює іншу;

3) включення (include) – показує, що для виконання певної дії обов’язковим є виконання іншої.

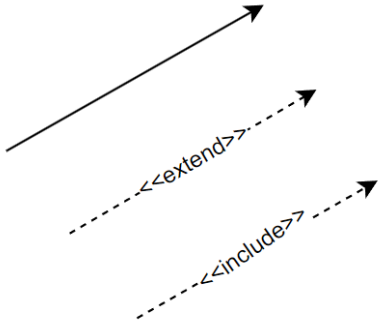


Рисунок 2.4 – Позначення зв’язків

На рисунку 2.5 зображена Use Case діаграма для додатку генерування часових рядів, що імітують навантаження в системах мікропослуг.

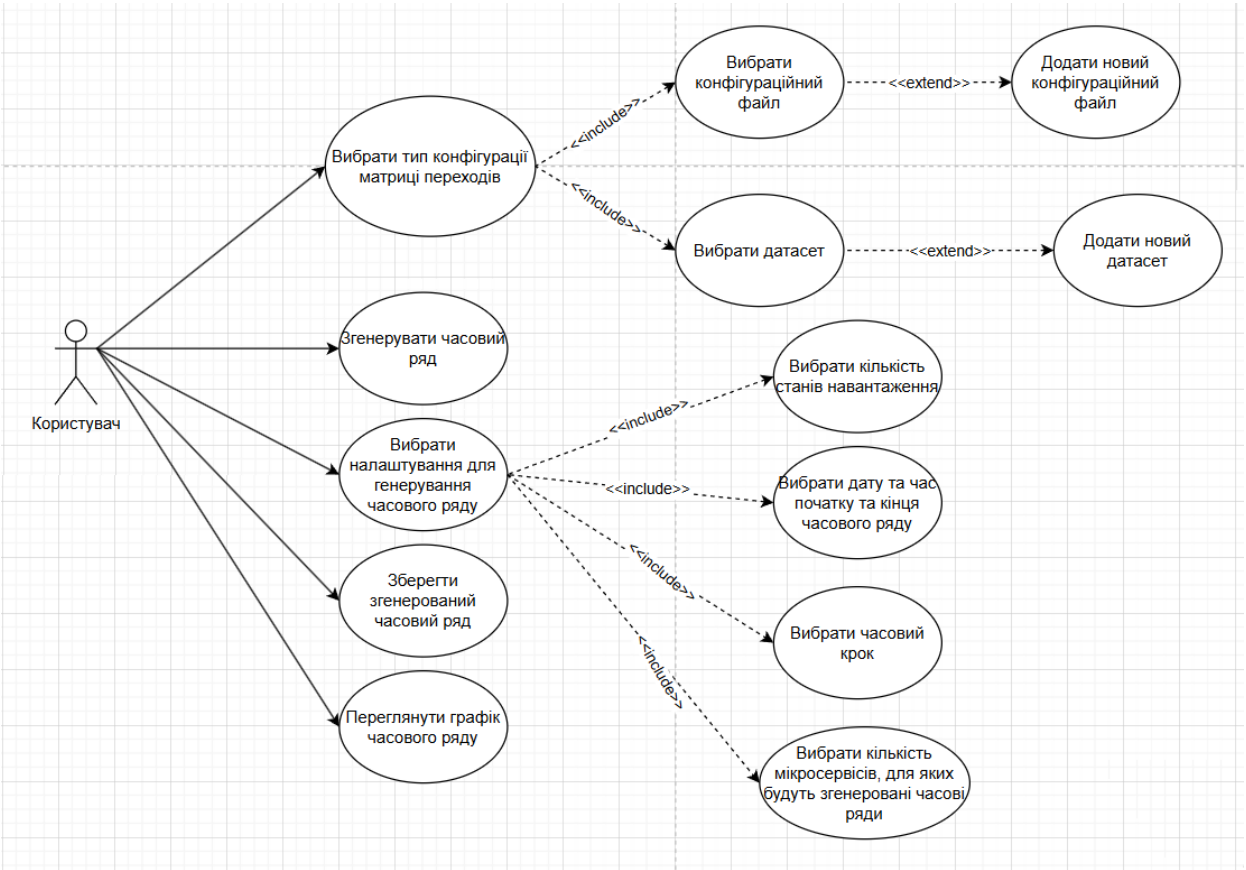


Рисунок 2.5 – Діаграма прецедентів

2.2.2 DFD diagram

Діаграма потоків даних або DFD (Data Flow Diagram) – це методологія графічного структурного аналізу, що описує зовнішні стосовно системи джерела та адресати даних, логічні функції, потоки даних та сховища даних, до яких здійснюється доступ[16]. Діаграма наочно демонструє шляхи, якими циркулюють дані всередині проектованої інформаційної системи, а також між шляхами проходження інформації між системою і зовнішнім світом.

Складовими такого типу UML діаграм є:

- процеси;
- сховища даних;
- зовнішні сутності;
- потоки даних.

Процеси виконують схожу функцію до варіантів використання Use Case діаграм, вони позначають процеси, які обробляють інформацію в системі. Позначаються такі елементи прямокутниками із закругленими кутами з назвою процесу всередині (рис. 2.6).

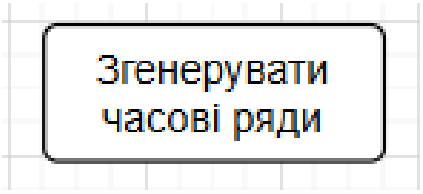


Рисунок 2.6 – Позначення процесів на діаграмах потоків даних

Сховища даних відображають контейнери для збереження інформації в системі. Як правило, сховище даних – це окрема таблиця в базі даних. Дані елементи на діаграмі позначаються прямокутниками без бічних граней з назвою сховища всередині (рис. 2.7).

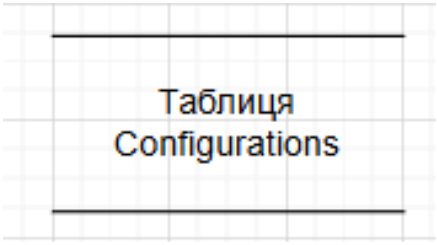


Рисунок 2.7 – Позначення сховища даних

Зовнішні сутності – це елементи, які не входять до складу системи, проте є джерелом, або отримувачем інформації після її обробки системою. Зовнішні сутності відіграють схожу роль до акторів в діаграмах варіантів використання. Позначаються вони звичайними прямокутниками із зазначеною назвою сутності (рис. 2.8).

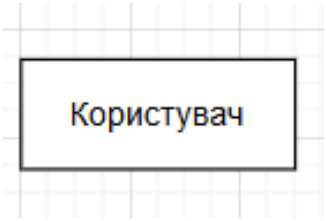


Рисунок 2.8 – Позначення зовнішньої сутності

Потоки даних відображають інформацію, яка передається від одного елемента діаграми до іншого. Вони показують, як елементи системи обмінюються даними між собою, та позначаються стрілками з назвою даних, що передаються (рис. 2.9).

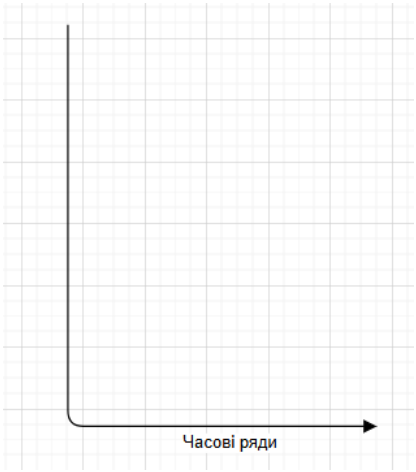


Рисунок 2.9 – Позначення потоку даних

Діаграма потоків даних для розроблюваного додатку зображена на рисунку 2.10.

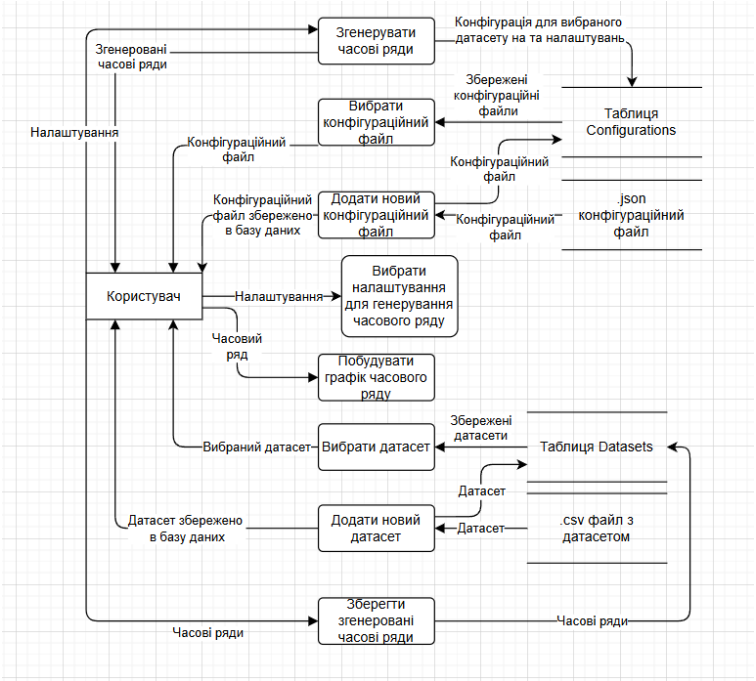


Рисунок 2.10 – DFD діаграма

2.2.3 IDEF0 diagram

IDEF0 – методологія функціонального моделювання, яка використовується для створення функціональної моделі, що відображає структуру і функції системи, а також потоки інформації і матеріальних об'єктів, що зв'язують ці функції[17]. Діаграми IDEF0 дозволяють більш детально показати як функції системи взаємодіють між собою: які дані вони відправляють, послідовність їх виконання, механізми, що підтримують їх роботу та елементи, які накладають обмеження чи правила на роботу функцій.

Ці діаграми складаються з таких елементів:

- блоки – представляють функції або процеси системи;
- входи – інформація, яка надходить до блоку;
- виходи – інформація, яку надсилає блок;

- управління – обмеження, правила чи стандарти, які регулюють роботу функцій системи;
- механізми – елементи, які підтримують виконання функцій (наприклад, сутності бази даних).

Блоки діаграми зображуються прямокутниками з назвою процесу, до якого під’єднані стрілки:

- входи – з лівої сторони;
- виходи – з правої сторони;
- управління – зверху;
- механізми – знизу.

На рисунку 2.11 зображено діаграму IDEF0 для головного процесу системи – генерування часового ряду.

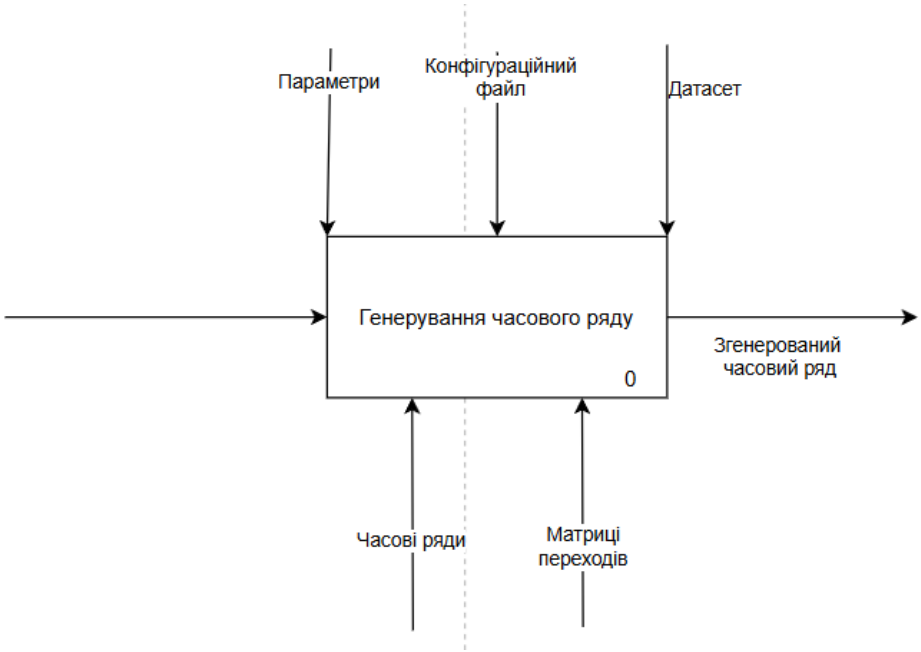


Рисунок 2.11 – Діаграма IDEF0

Нотація IDEF0 також дозволяє декомпозувати процеси, тобто описати їх більш детально та розбити на менші підпроцеси. На рисунку 2.12 показано декомпозицію головного процесу системи.

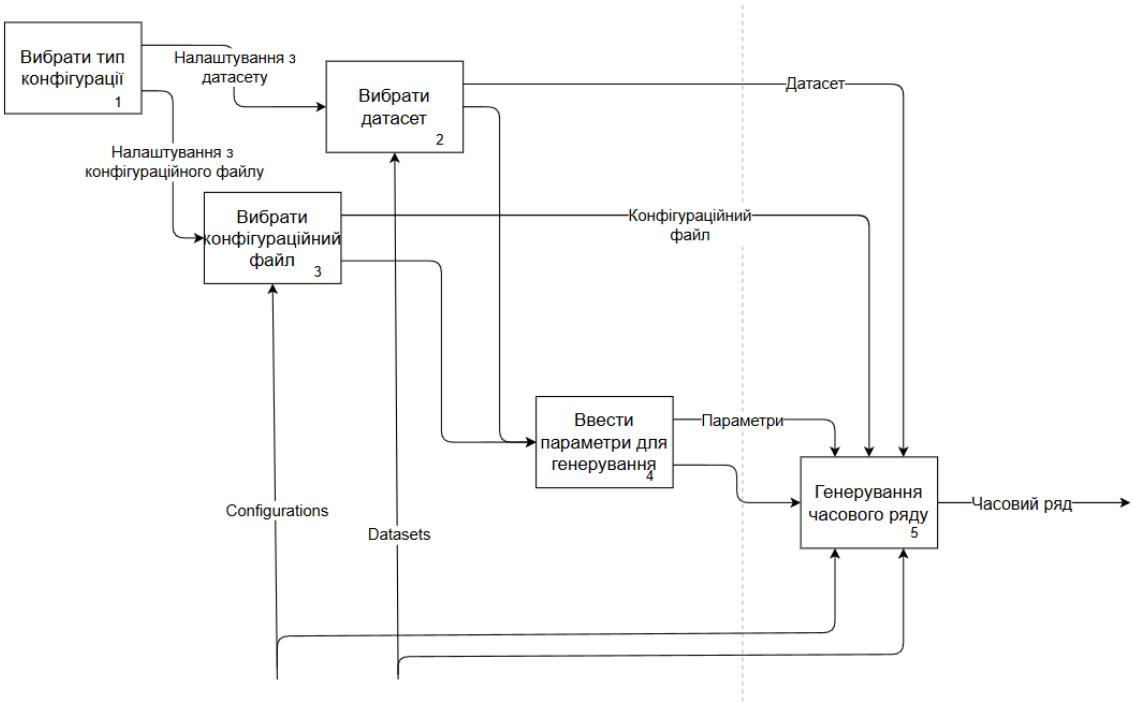


Рисунок 2.12 – Декомпозиція головного процесу в нотації IDEF0

Висновки до розділу 2

У другому розділі проаналізовано підходи до розв’язання поставленої задачі. Розглянуто такі існуючі методи генерування часових рядів як ARIMA, Random Forest, RNN.

Для реалізації поставленої задачі обрано метод ланцюгів Маркова, виділені його переваги, зокрема реалізацію незалежності наступних значень часового ряду від попередніх. Також цей підхід дає можливість генерувати часові ряди без наявності наборів вхідних даних і дозволяє генерувати різні дані при однакових налаштуваннях.

Було висунуто вимоги до розроблюваного програмного продукту та на їх основі побудовано архітектуру системи. Основні способи використання системи було описано за допомогою Use Case діаграми, а загальну роботу системи було відображено на DFD та IDEF0 діаграмах.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

Для реалізації алгоритму генерування часових рядів, що імітують навантаження в мікросервісних системах, було вирішено використовувати Марківські процеси, адже навантаження на сервіси може змінюватися непередбачувано, незалежно від попередніх значень навантаження, що Марківські процеси дозволяють відтворити. Також використання ланцюгів Маркова дозволяє генерувати дані без існуючих датасетів, вручну налаштувавши матрицю переходів.

Систему було поділено на 6 модулів:

1) load_data.py – модуль завантаження датасетів з .csv файлів та конфігураційних файлів типу .json;
2) plot.py – модуль для побудови графіків часових рядів;
3) db.py – модуль, що відповідає за роботу з базою даних;
4) workload_generator.py – модуль, що містить алгоритм генерування часових рядів на основі ланцюгів Маркова;
5) UI.py – модуль, що реалізує інтерфейс програми;
6) main.py – модуль, що налаштовує базу даних та запускає програму.

Схему взаємодії модулів системи між собою можна побачити на рисунку 3.1.

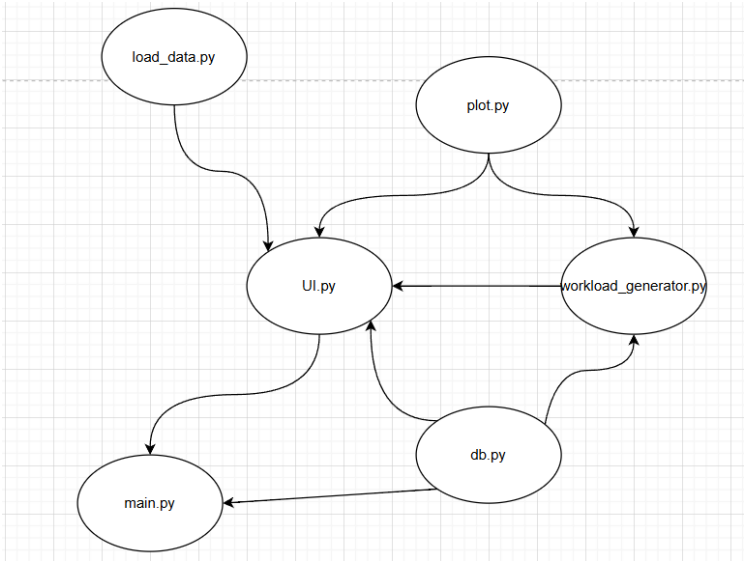


Рисунок 3.1 – Схема взаємодії модулів системи

3.1 База даних

База даних розроблюваної системи містить дві таблиці: Datasets, яка зберігає дані про часові ряди, та Configurations, яка зберігає матриці переходів.

Таблиця Datasets містить наступні поля:

- id – ідентифікатор часового ряду;
- service_name – назва, за якого можна знайти часовий ряд;
- metric_type – тип метрики (CPU, RAM, CHANNEL);
- date – дата запису про навантаження;
- time – час запису про навантаження;
- value – значення навантаження.

Таблиця Configurations має такі поля:

- id – ідентифікатор матриці;
- states – кількість станів навантаження;
- is_auto_generated – матриця була згенерована автоматично (після її налаштування з датасету), чи вручну (при додаванні з файлового провідника конфігураційного файлу);
- name – назва конфігураційного файлу;
- dataset_id – ідентифікатор часового ряду, для якого була згенерована матриця;
- from_state – початковий стан навантаження;
- to_state – наступний стан навантаження;
- CPU – ймовірність переходу для навантаження процесора;
- RAM – ймовірність переходу для використаної пам’яті;
- CHANNEL – ймовірність переходу для використаного канального ресурсу.

Концептуальну модель бази даних можна побачити на рисунку 3.2.

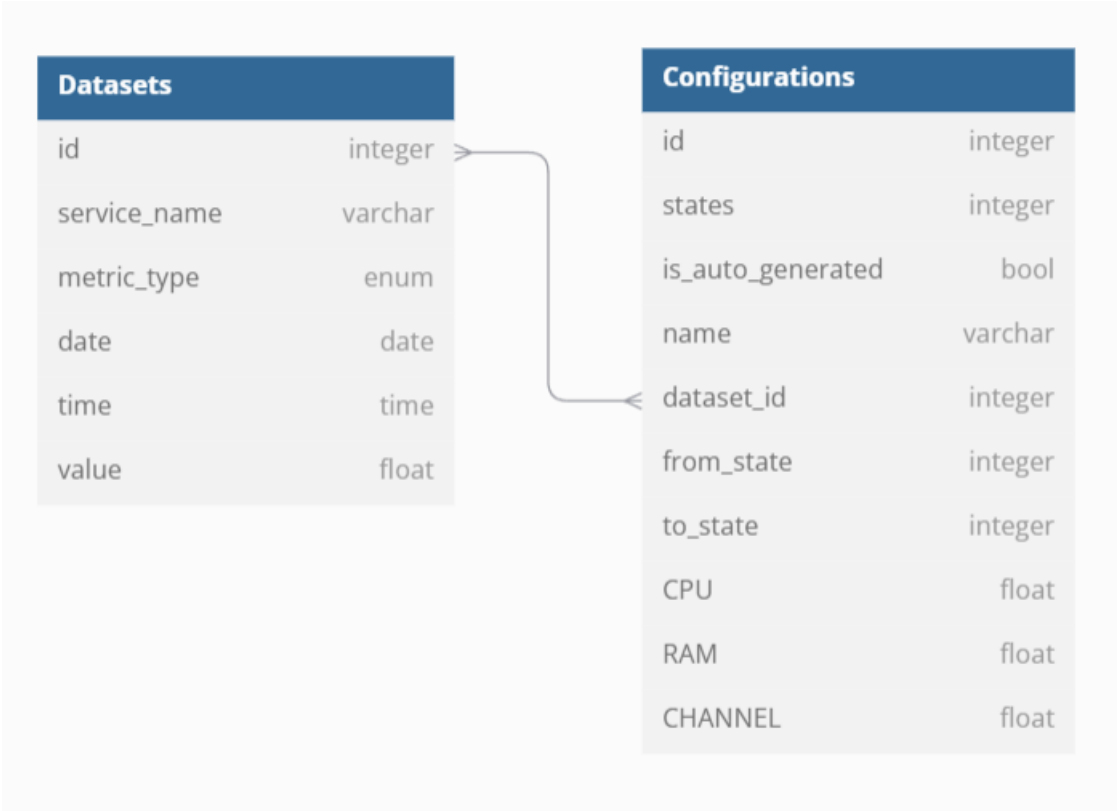


Рисунок 3.2 – Концептуальна модель бази даних

3.2 Генератор часових рядів

Ланцюги Маркова в даній роботі використовуються для реалізації переходів між станами навантаження на мікросервіси. Кількість станів навантаження задається користувачем, та означає кількість частин, на які буде поділено стовідсоткове навантаження. Наприклад, якщо задається чотири стани навантаження, то 100% навантаження ділиться на чотири стани:

1) низький – навантаження в межах 0-25%;
2) середній – навантаження в межах 26-50%;
3) високий – навантаження в межах 51-75%;
4) піковий – навантаження в межах 76-100%.

Таким чином формується матриця переходів, яка містить ймовірності переходу системи з одного стану в інший. Оскільки система має чотири стани навантаження, то така матриця матиме розмір 4x4 (рис. 3.3).

$$P = \begin{pmatrix} 0.4 & 0.3 & 0.2 & 0.1 \\ 0.2 & 0.5 & 0.2 & 0.1 \\ 0.1 & 0.3 & 0.4 & 0.2 \\ 0.1 & 0.1 & 0.4 & 0.4 \end{pmatrix}$$

Рисунок 3.3 – Приклад матриці переходів між станами навантаження

Кожен рядок такої матриці відображає стан, в якому в певний момент часу перебуває система, а стовпчики – стани, в які система може перейти з поточного. Сума ймовірностей в кожному рядку повинна бути рівною одиниці.

Програмний продукт дозволяє налаштовувати матрицю переходів двома способами: з конфігураційного файлу, який містить матриці переходів для кожного параметру (навантаження на процесор, кількість спожитої пам’яті та канального ресурсу), або ж за допомогою датасету, який містить дані про навантаження для тих же параметрів в певний момент часу.

3.2.1 Налаштування з конфігураційних файлів

Конфігураційними файлами виступають файли з розширенням .json, які містять матриці переходів між станами навантаження для параметрів серверу (рис. 3.4).

middle_config.json

```
{
  "States": 4,
  "MarkovChain": {
    "CPU": [[0.2, 0.6, 0.17, 0.03], [0.19, 0.7, 0.1, 0.01], [0.09, 0.6, 0.3, 0.01], [0.05, 0.4, 0.45, 0.1]],
    "RAM": [[0.2, 0.6, 0.17, 0.03], [0.19, 0.7, 0.1, 0.01], [0.09, 0.6, 0.3, 0.01], [0.05, 0.4, 0.45, 0.1]],
    "CHANNEL": [[0.2, 0.6, 0.17, 0.03], [0.19, 0.7, 0.1, 0.01], [0.09, 0.6, 0.3, 0.01], [0.05, 0.4, 0.45, 0.1]]
  }
}
```

Рисунок 3.4 – Приклад конфігураційного файлу

Для налаштування матриці за допомогою конфігураційного файлу система перевіряє цей файл на коректність за такими правилами:

- файл повинен містити матриці для трьох параметрів серверу (навантаження на процесор, кількість використаної пам’яті та канального ресурсу);
- кількість рядків в матрицях повинна бути однаковою;
- кількість стовпчиків в матрицях повинна бути однаковою;
- сума ймовірностей переходу з певного стану навантаження в інші повинна дорівнювати одиниці;
- ймовірність повинна бути в межах (0, 1).

Якщо всі ці правила були дотримані, то система налаштовує матрицю переходів за заданими матрицями у файлі.

3.2.2 Налаштування за допомогою датасету

Матриця переходів також може бути налаштована на основі заданих даних про навантаження на компоненти серверу в певний момент часу. Для цього датасет повинен мати наступні дані (рис. 3.5):

- дата;
- час;
- відсоток навантаження процесору;
- відсоток використаної пам’яті;
- відсоток використаного канального ресурсу.

3.csv

CSV

	Date	Time	CPU	RAM	CHANNEL
1	2013-08-12	16:40:46	3.3666666666666667	5.066649119059244	12.0
2	2013-08-12	16:45:46	3.7	5.266650517781576	12.333333333333336
3	2013-08-12	16:50:46	3.4	6.266641616821289	12.266666666666667
4	2013-08-12	16:55:46	3.333333333333333	4.1333166758219395	11.533333333333333
5	2013-08-12	17:00:46	3.533333333333333	4.866650899251302	10.6
6	2013-08-12	17:05:46	3.166666666666667	8.133316040039062	11.6
7	2013-08-12	17:10:46	3.3666666666666667	5.333305994669597	11.266666666666667
8	2013-08-12	17:15:46	3.666666666666667	4.199981689453125	11.4
9	2013-08-12	17:20:46	3.333333333333333	5.733308792114258	11.0
10	2013-08-12	17:25:46	3.1	6.533323923746743	10.933333333333335

Рисунок 3.5 – Приклад датасету

На основі цих даних визначається стан навантаження в певний момент часу. Далі для кожного стану підраховується кількість переходів в кожен інший стан навантаження, в результаті чого отримується матриця, яка містить дані про кількість переходів між станами системи (рис. 3.6).

$$P_{count} = \begin{pmatrix} 100 & 75 & 50 & 25 \\ 40 & 100 & 40 & 20 \\ 10 & 30 & 40 & 20 \\ 8 & 8 & 32 & 32 \end{pmatrix}$$

Рисунок 3.6 – Матриця кількостей переходів між станами навантаження

В результаті додавання всіх елементів рядків описаної вище матриці, отримується матриця, яка містить загальні кількості переходів, виконаних з певного стану навантаження (рис. 3.7).

$$P_{sum} = \begin{pmatrix} 250 \\ 200 \\ 100 \\ 80 \end{pmatrix}$$

Рисунок 3.7 – Матриця виконаних переходів з певного стану навантаження

Обрахування елементів матриці P_{sum} описане у формулі (3.1).

$$P_{sum_{i1}} = \sum_{j=1}^{n} P_{count_{ij}} \tag{3.1}$$

де P_{sum} – матриця кількостей переходів з певного стану;

P_{count} – матриця кількостей переходів;

i – номер рядка матриці;

j – номер стовпчика матриці;

n – розмір матриці.

Після цього для кожного рядка матриці P_{count} потрібно знайти коефіцієнт, на який потрібно помножити кожен елемент, щоб отримати ймовірності переходів

так, щоб сума кожного рядку отриманої матриці дорівнювала одиниці. Зробити це можна шляхом ділення одиниці на елементи матриці P_{sum}. В результаті отримується матриця з коефіцієнтами (рис. 3.8).

$$P_{coef} = \begin{pmatrix} 0.004 \\ 0.005 \\ 0.01 \\ 0.0125 \end{pmatrix}$$

Рисунок 3.8 – Матриця коефіцієнтів

Обрахування елементів матриці P_{coef} описане у формулі (3.1).

$$P_{coef_{i1}} = \frac{1}{P_{sum_{i1}}} \tag{3.2}$$

де P_{coef} – матриця коефіцієнтів;

P_{sum} – матриця кількостей переходів з певного стану;

i – номер рядка матриць.

В результаті множення значення рядка матриці коефіцієнтів на елементи відповідного рядка матриці P_{count} отримується потрібна матриця ймовірностей переходів з одного стану системи в інший (рис. 3.3). Обрахування елементів матриці P_{sum} описане у формулі (3.1).

$$P_{ij} = P_{count_{ij}} * P_{coef_{i1}} \tag{3.3}$$

де P – матриця переходів;

P_{count} – матриця кількостей переходів;

P_{coef} – матриця коефіцієнтів;

i – номер рядка матриці;

j – номер стовпчика матриці.

Отже, налаштування матриці переходів на основі датасету включає в себе такі етапи:

- перевірка файлу датасету на коректність;
- створення матриці, яка містить кількості переходів з одного стану навантаження в інший;
- створення матриці, яка містить загальні кількості переходів з певного стану навантаження;
- знаходження коефіцієнту для кожного стану;
- створення матриці переходів між станами навантаження.

Алгоритм налаштування можна побачити на рисунку 3.9.

Рисунок 3.9 – Алгоритм налаштування матриці переходів на основі датасету

3.3 Розробка інтерфейсу користувача

Інтерфейс користувача розроблявся за допомогою фреймворку PyQt, який є оболонкою бібліотеки Qt на мові програмування Python. PyQt є одним з найпопулярніших засобів Python для роботи з кросплатформним C++ фреймворком Qt[18].

PyQt широко використовується для створення інтерфейсу користувача на мові програмування Python через широкий вибір компонент (кнопки, лейаути, величезна різноманітність полів для вводу, таблиці, тощо), потужність фреймворку Qt, та зручності його використання на мові програмування Python.

Для розробки інтерфейсу було створено .ui файл, який описує головне вікно програми. Цей файл завантажується вбудованою функцією loadUi фреймворку PyQt в модулі UI.py, яка перетворює цей файл у повноцінний інтерфейс.

Створене головне вікно програми можна побачити на рисунку 3.10.

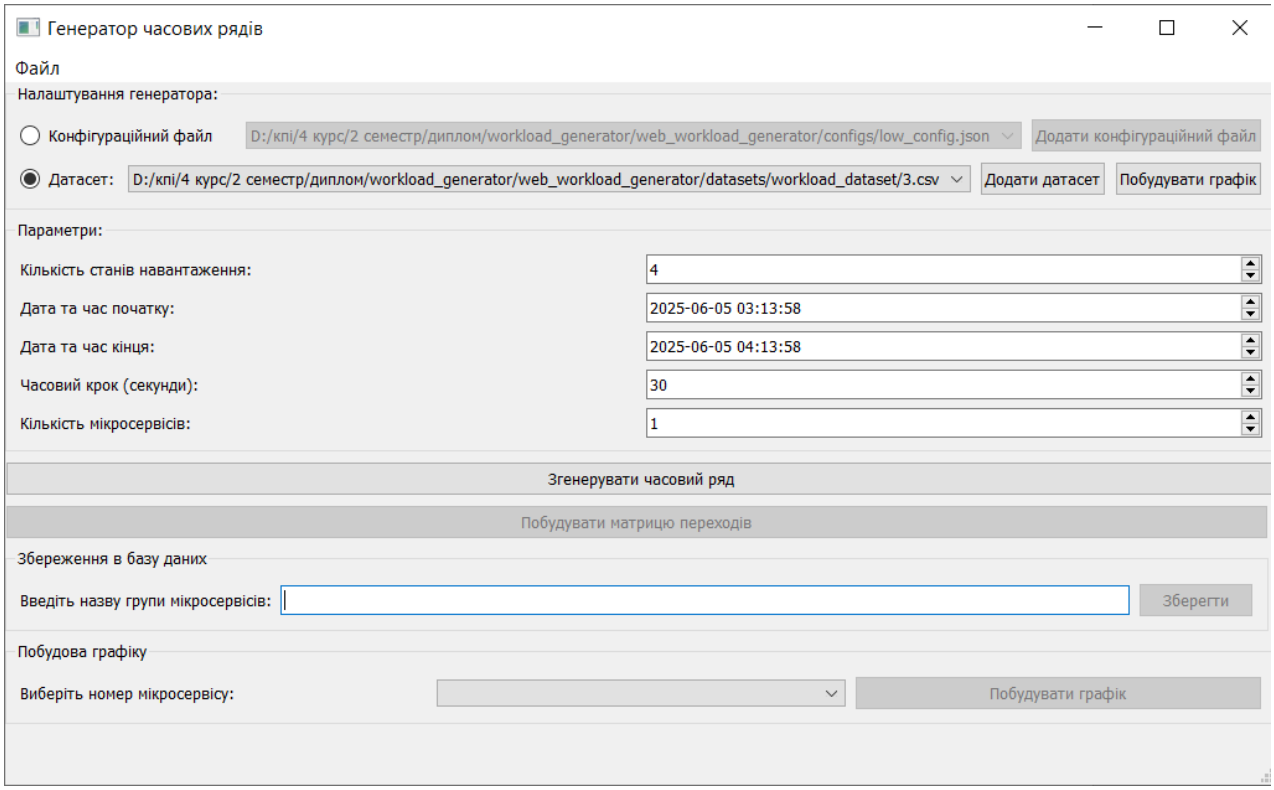


Рисунок 3.10 – Створене головне вікно програми

3.4 Використані програмні засоби

3.4.1 IDE PyCharm

Процес написання програмного коду для серверної частини додатку відбувався в IDE PyCharm. Це інтегроване середовище розробки, яке було створене під мову Python, яке є дуже популярним через свій широкий функціонал і зручність у розробці ПЗ. PyCharm розповсюджується як у безкоштовній версії (Community), так і в платній версії (Professional), яку студент може отримати безкоштовно. Даний IDE містить редактор коду, дебагер та інструменти для роботи з Git.

Інструмент для дебагінгу є особливо зручним, адже він дуже допомагає, коли програма працює не так, як повинна це робити. Якщо не вдається самотужки знайти проблему в коді, то за допомогою дебагера можна легко знайти помилку. Для того щоб почати сам процес, потрібно в коді поставити точку зупинки. Її можна поставити перед фрагментом коду, в якому є підозра на помилку, або ж поставити її на початку виконання програми, щоб пробігтися по всьому коду. Під час дебагінгу відбувається покроковий рух по коду, під час якого показуються поточне значення змінних, результати роботи функцій. За допомогою аналізу цих значень можна точно визначити, в чому проблема роботи програми.

Середовище має приємний та зручний інтерфейс, підсвічування коду, яке можна налаштувати під свої потреби та смак. Також реалізовано підказки щодо доповнення коду, що є дуже корисним та зручним, адже це допомагає, наприклад, побачити які поля та методи має об’єкт класу, не повертаючись до фрагменту коду з реалізацією самого класу. Також PyCharm, як і всі продукти JetBrains пропонує у своїх IDE доповнювання коду за допомогою штучного інтелекту, що іноді може значно пришвидшити процес написання, адже ШІ робить все за Вас, проте не варто на це завжди полягатися, адже штучний інтелект може досить часто помилятися. Інтерфейс даного IDE зображено на рисунку 3.11.

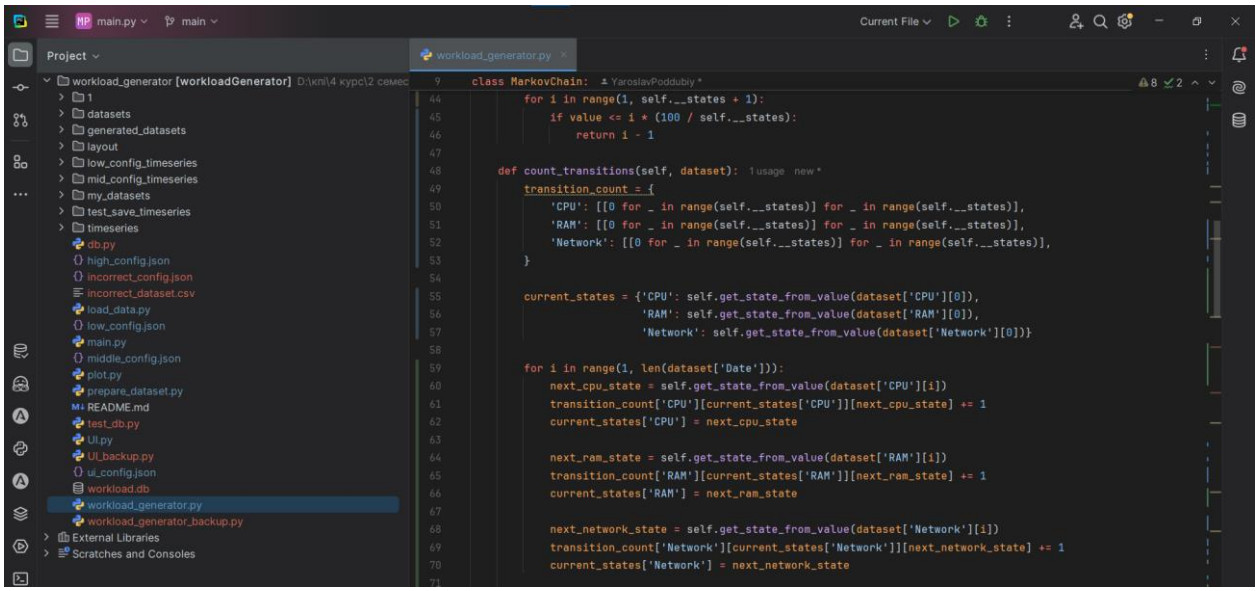


Рисунок 3.11 – Інтерфейс середовища розробки PyCharm

3.4.2 Qt Designer

Qt Designer – це додаток, який дозволяє створювати .ui файли для PyQt за допомогою графічного інтерфейсу. Він дозволяє позбутися ручного набору коду для створення графічного інтерфейсу з використанням PyQt, що значно пришвидшує його розробку.

Цей засіб дозволяє додавати елементи до інтерфейсу шляхом їх перетягування з каталогу елементів до вікна розроблюваного графічного інтерфейсу, налаштовувати шрифти, тексти, встановлювати підказки, налаштовувати різні види розміщення елементів (лейаути). Також він дозволяє налаштовувати сигнали та дії.

Сигналами можуть бути натискання кнопки, перемикання перемикачів, вибір іншого елементу зі списку. Ці сигнали тригерять дії, до яких вони прив’язані. Наприклад, перемикання перемикача може активовувати чи деактивовувати певні взаємодії з інтерфейсом. Інтерфейс даного додатку зображений на рисунку 3.12.

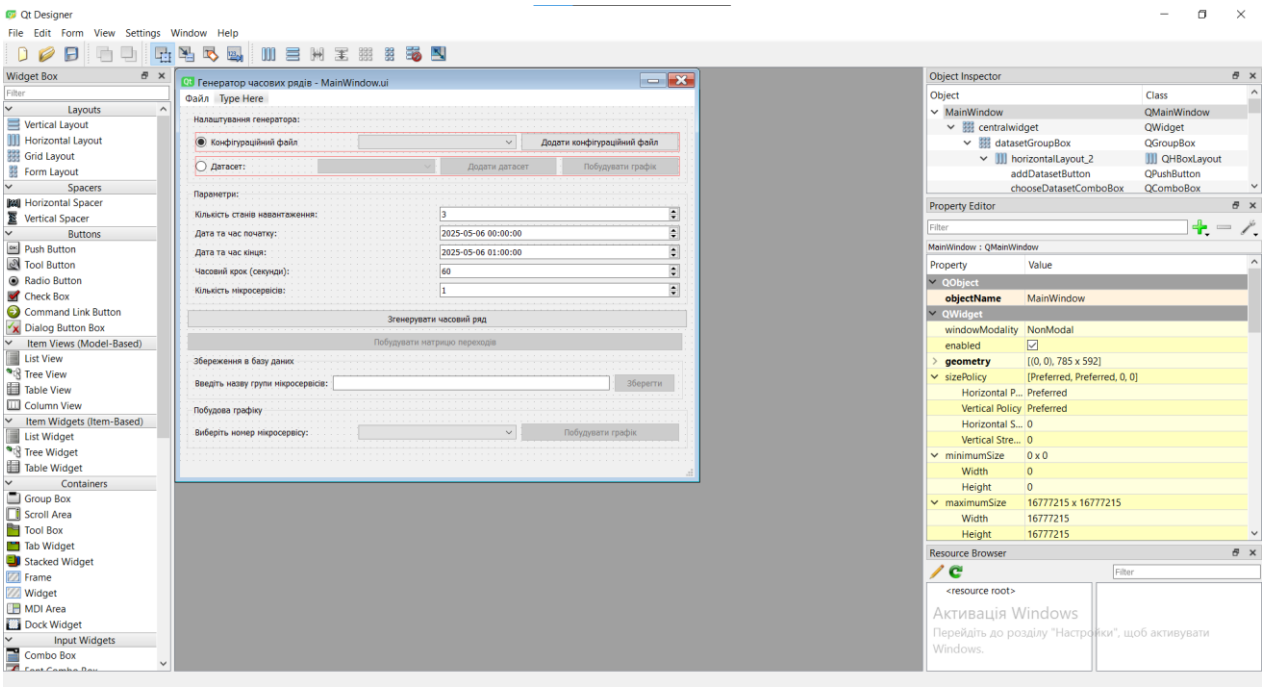


Рисунок 3.12 – Інтерфейс Qt Designer

3.5 Тестування системи

Для тестування системи проводилося ручне тестування. Перевірялася робота системи при введені користувачем некоректних даних, таких як:

- конфігураційний файл;
- файл з датасетом;
- кількість станів навантаження;
- часовий крок;
- кількість мікросервісів;
- назва групи мікросервісів.

Для таких параметрів як кількість станів навантаження, часовий крок і кількість мікросервісів були встановлені обмеження зі сторони інтерфейсу користувача. Поля вводу цих параметрів обмежують їхні значення відповідно до таблиці 3.1.

Таблиця 3.1 – Обмеження значень параметрів

Назва параметру	Мінімальне значення	Максимальне значення
Кількість станів навантаження	3	20
Часовий крок (секунди)	1	600
Кількість мікросервісів	1	20

Назва групи мікросервісів повинна бути унікальною, тому користувач повинен ввести таку назву, яка ще не міститься в базі даних. У разі, якщо користувач введе вже існуючу назву групи мікросервісів, він отримає про це попередження (рис. 3.13) та дані не будуть збережені в БД.

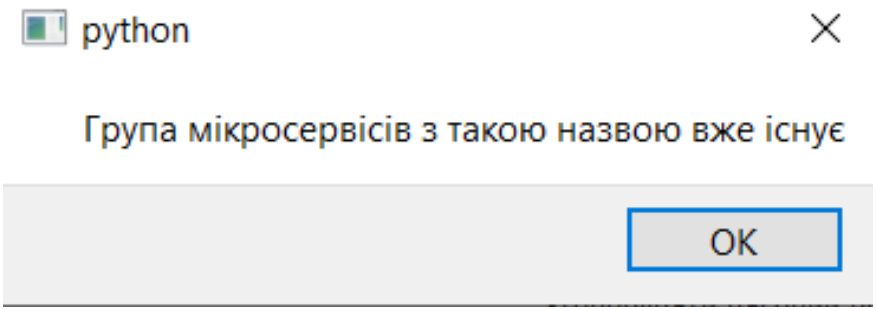


Рисунок 3.13 – Попередження про існуючу назву

Якщо користувач захоче додати свій конфігураційний файл до БД, цей файл повинен задовольняти наступні умови:

- містити поля “States” і “MarkovChain”;
- значення “States” має співпадати з розмірами матриць, які містяться в полі “MarkovChain”;
- матриці повинні бути квадратними;
- сума ймовірностей кожного з рядків матриць повинна бути рівною одиниці;
- значення в матрицях повинні лежати в межах (0, 1).

У випадку відсутності одного з полів “States” чи “MarkovChain” відображається попередження, зображене на рисунку 3.14.

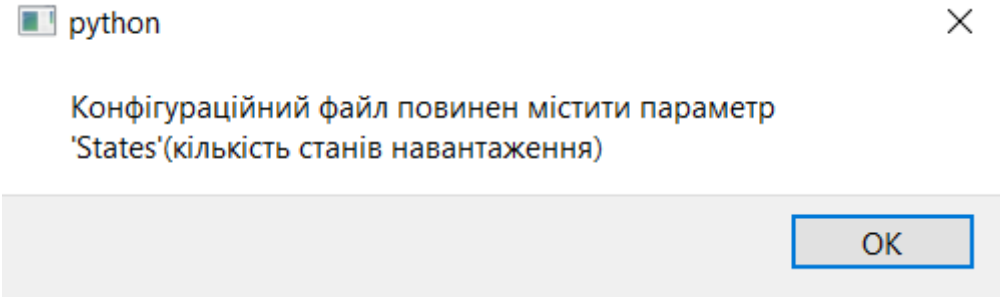


Рисунок 3.14 – Помилка при відсутності поля “States”

Якщо значення параметру “States” не співпадає з кількістю рядків матриць для кожного параметру навантаження, то користувач отримає помилку (рис. 3.15).

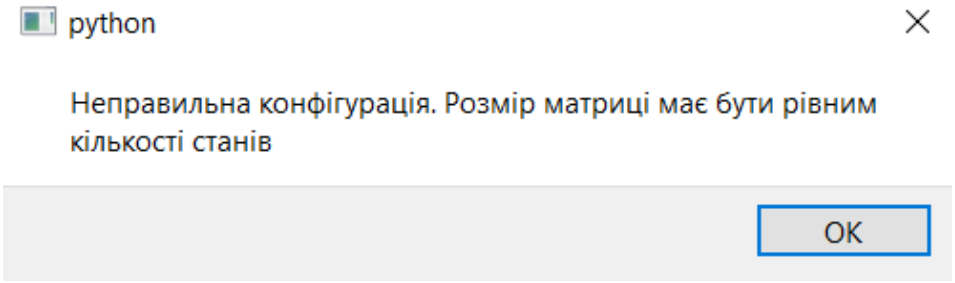


Рисунок 3.15 – Помилка при неправильному розмірі матриці

При додаванні конфігураційного файлу, в якому хоч одна з матриць не є квадратною (кількість рядків не дорівнює кількості стовпців), буде показане вікно з помилкою (рис. 3.16).

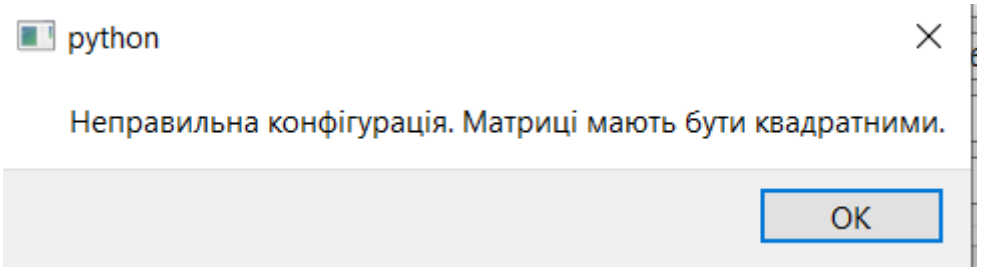


Рисунок 3.16 – Помилка при не квадратній матриці

Якщо сума ймовірностей в рядку матриці не дорівнює одиниці, на екран буде виведено повідомлення про помилку (рис. 3.17).

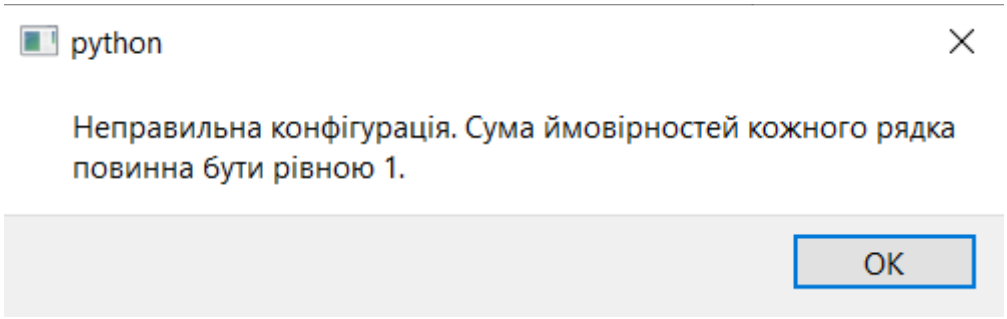


Рисунок 3.17 – Помилка, якщо сума ймовірностей в рядку не дорівнює 1

Якщо сума ймовірностей рівна одиниці, але хоч одне значення виходить за межі (0, 1), то користувач отримає відповідне повідомлення з помилкою (рис. 3.18).

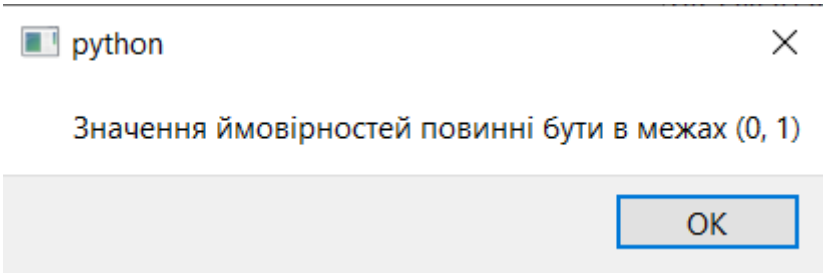


Рисунок 3.18 – Помилка при недопустимих значеннях ймовірностей

Для того, щоб додати до БД файл з набором даних, на основі яких будуватиметься матриця переходів та генеруватимуться часові ряди, такий файл повинен:

- містити параметри “Date”, “Time”, “CPU”, “RAM” і “CHANNEL”;
- дата та час повинні бути записані в правильному форматі;
- параметри “CPU”, “RAM” і “CHANNEL” повинні бути в числовому форматі;
- значення параметрів “CPU”, “RAM” і “CHANNEL” повинні бути в межах [0, 100].

Якщо перший рядок файлу не містить назви параметрів “Date”, “Time”, “CPU”, “RAM” і “CHANNEL”, то користувач отримає відповідне повідомлення про помилку (рис. 3.19).

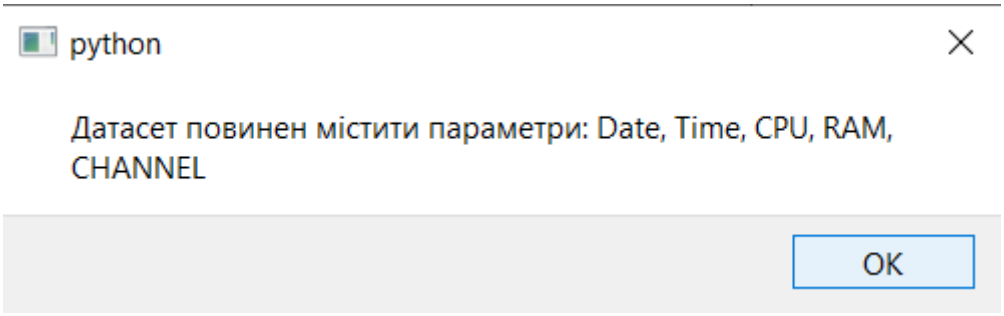


Рисунок 3.19 – Помилка при неправильних параметрах датасету

Якщо дата та час, були введені не в правильному форматі, на екрані з’явиться помилка, яка повідомляє про це та вказує формат, в якому повинні бути введені час та дата (рис. 3.20).

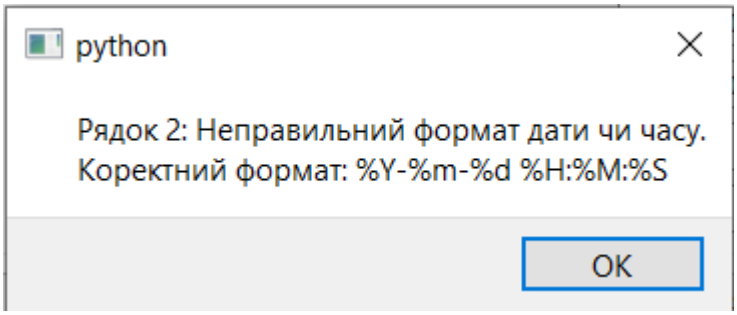


Рисунок 3.20 – Помилка про неправильний формат дати чи часу

При неправильному форматі значень параметрів навантаження “CPU”, “RAM” чи “CHANNEL” буде виведена помилка, зображена на рисунку 3.21.

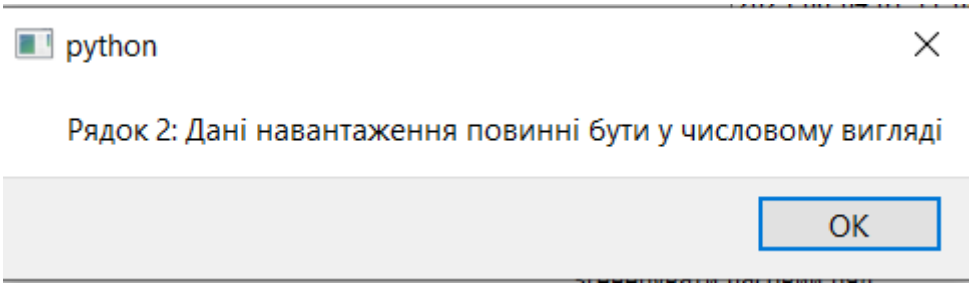


Рисунок 3.21 – Помилка про неправильний формат значень навантаження

Якщо ж усі значення навантаження були введені в правильному числовому форматі, але їх значення виходять за межі [0, 100], то користувач отримає помилку, яка сповістить про вихід за межі допустимих значень (рис. 3.22).

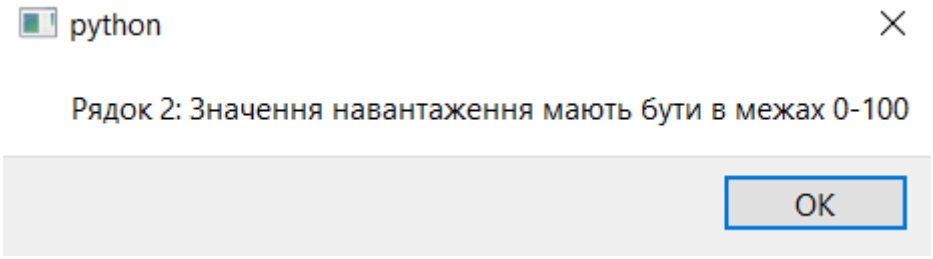


Рисунок 3.22 – Помилка при виході значень навантаження за межі

Висновки до розділу 3

У третьому розділі було обрано технології та програмні засоби для реалізації програмного продукту.

Побудовано концептуальну модель бази даних. Створено дві таблиці для зберігання датасетів та матриць переходів: Datasets та Configurations.

Розроблено алгоритм для генерування часових рядів, що імітують навантаження систем мікросервісів на основі ланцюгів Маркова.

Проведено ручне тестування системи шляхом задання некоректних вхідних даних.

4 ІНТЕРФЕЙС КОРИСТУВАЧА

Інтерфейс користувача розроблявся за допомогою фреймворку PyQt, який є оболонкою бібліотеки Qt на мові програмування Python.

Головне вікно програми зображене на рисунку 4.1.

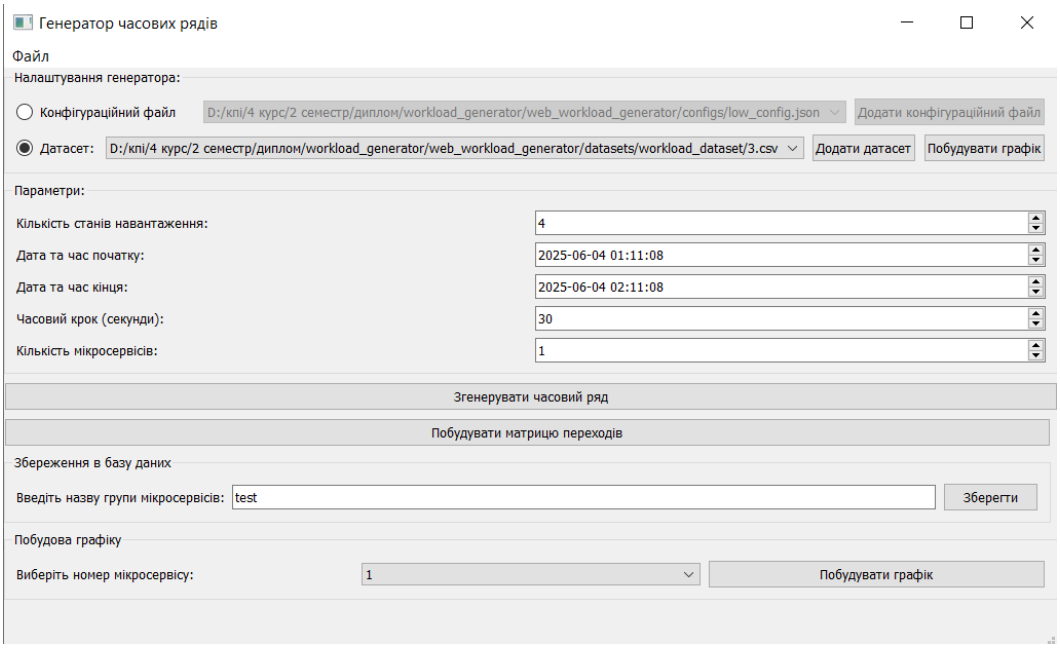


Рисунок 4.1 – Головне вікно програми

Тут користувач може:

- вибрати тип налаштування матриці переходів (за допомогою конфігураційного файлу або датасету);
- вибрати конфігураційний файл зі списку;
- додати новий конфігураційний файл з файлового провідника;
- вибрати датасет зі списку;
- додати новий датасет з файлового провідника;
- вибрати кількість станів навантаження;
- вибрати дату та час початку генерування даних;
- вибрати дату та час закінчення генерування даних;
- вибрати часовий крок, з яким генеруватиметься нове значення;

- вибрати кількість мікросервісів, для яких будуть генеруватися часові ряди;
- згенерувати часові ряди;
- вибрати номер мікросервісу, для якого будуватимуться графіки;
- переглянути графіки згенерованих часових рядів;
- переглянути графіки вхідних датасетів;
- зберегти згенеровані часові ряди в базу даних;
- зберегти згенеровані часові ряди у вигляді .csv файлів;
- вибрати назву для групи мікросервісів;
- переглянути матрицю переходів.

Загалом, для того, щоб згенерувати часові ряди, користувач повинен виконати алгоритм, зображений на рисунку 4.2.

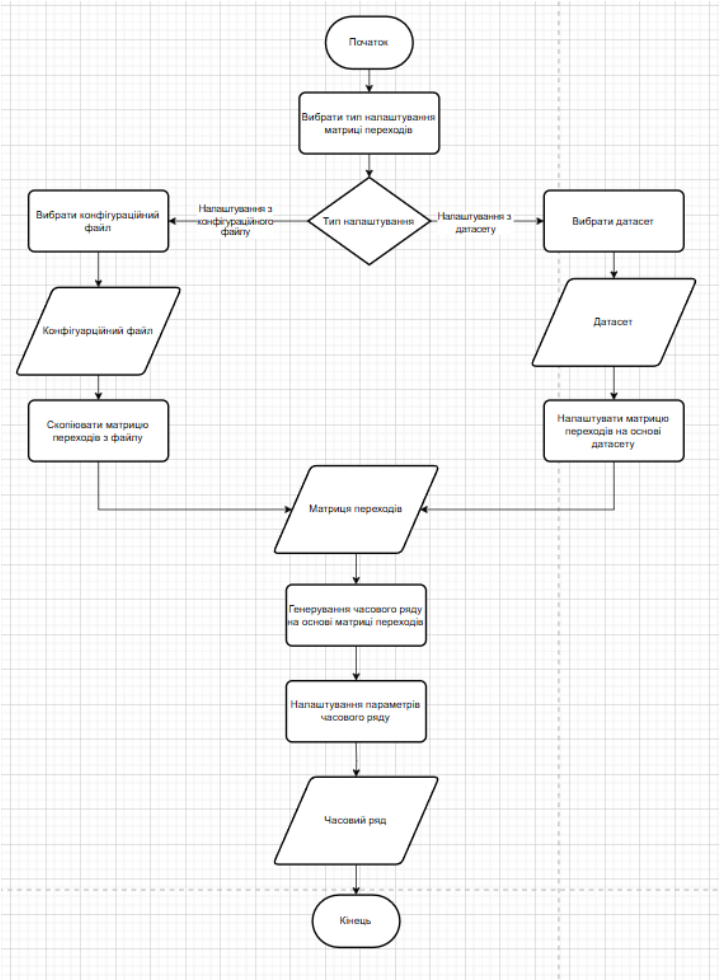


Рисунок 4.2 – Алгоритм для генерування часового ряду

Спочатку користувач обирає спосіб налаштування матриці переходів та вибирає зі списку доступних конфігураційний файл або ж датасет (рис. 4.3).

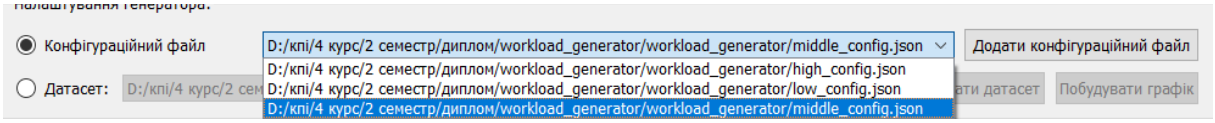


Рисунок 4.3 – Список доступних конфігураційних файлів

При наведенні курсору на поле вибору конфігураційного файлу вилізе підказка, яка міститиме інформацію про те, що таке конфігураційний файл і яку конструкцію він повинене містити (рис. 4.4).

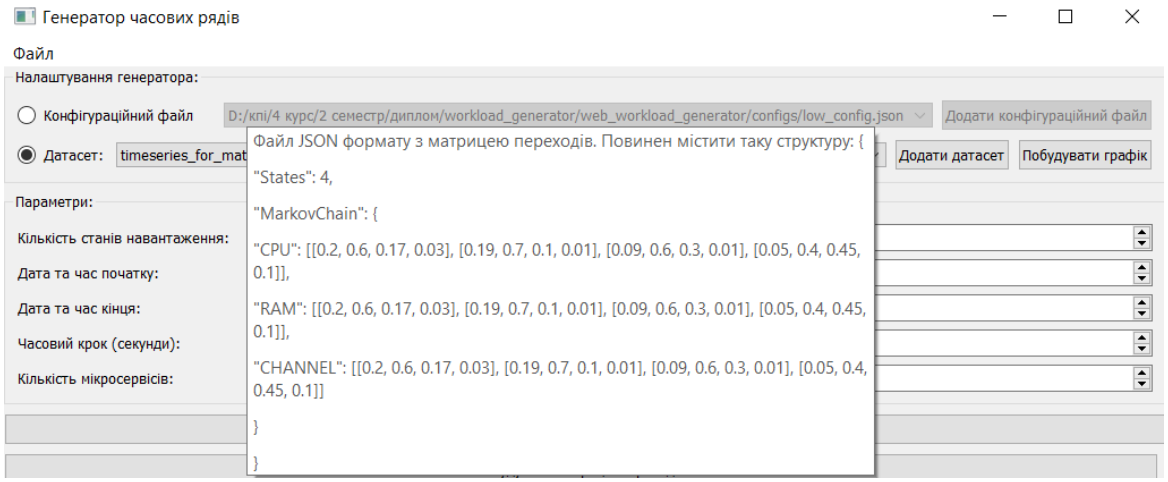


Рисунок 4.4 – Підказка для конфігураційного файлу

При наведенні курсору на поле вибору датасету вилізе підказка про те, що таке датасет, якого формату він повинен бути, які поля він повинен містити та в якому вигляді представляються значення (рис. 4.5).

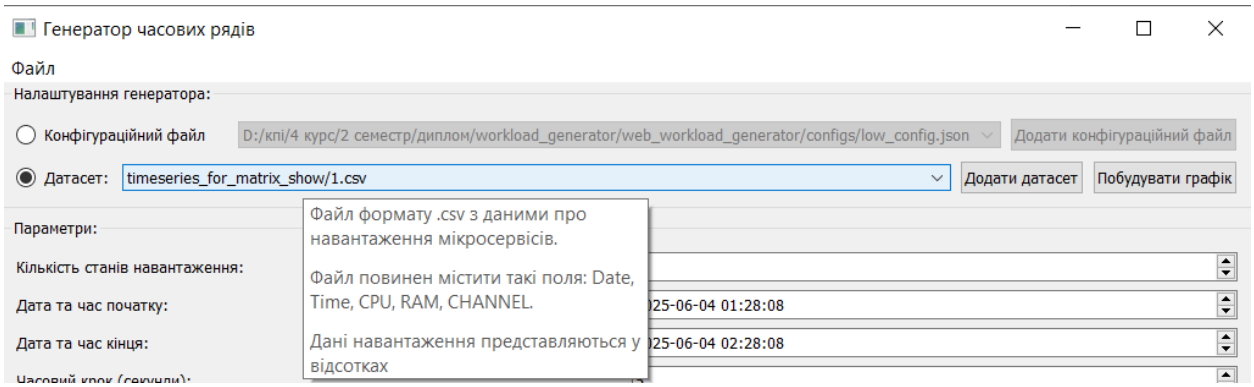


Рисунок 4.5 – Підказка для датасету

При наведенні курсору на поле вводу параметрів для налаштування генерування часових рядів також показуються підказки, які містять інформацію про те, за що відповідає кожен параметр, для того, щоб користувач міг правильно зрозуміти та вибрати потрібні йому налаштування(рис. 4.6).

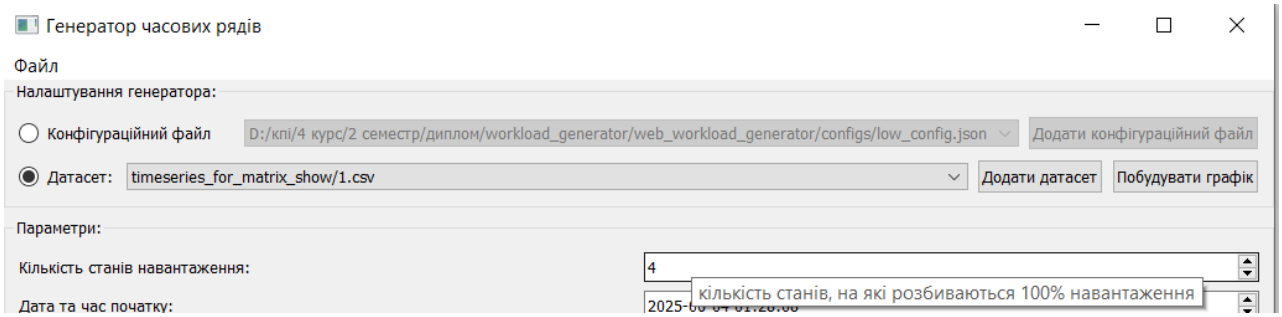


Рисунок 4.6 – Підказка для кількості станів навантаження

Також користувач може додати новий конфігураційний файл чи датасет з файлового провідника натиснувши на кнопку «Додати конфігураційний файл» або «Додати датасет» відповідно (рис. 4.7).

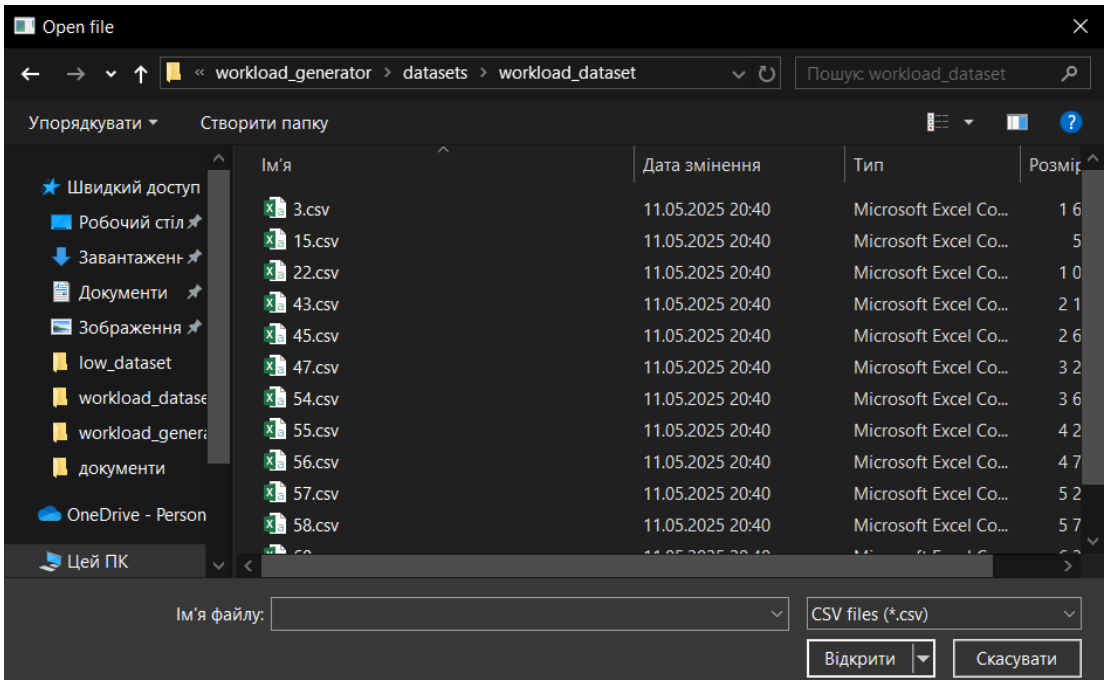


Рисунок 4.7 – Додавання нового датасету

Якщо користувач обрав тип налаштування з датасету, та вибрав потрібний набір даних з БД, то він може переглянути графік вибраного датасету, натиснувши на кнопку «Побудувати графік» в полі налаштування генератора (рис. 4.8).

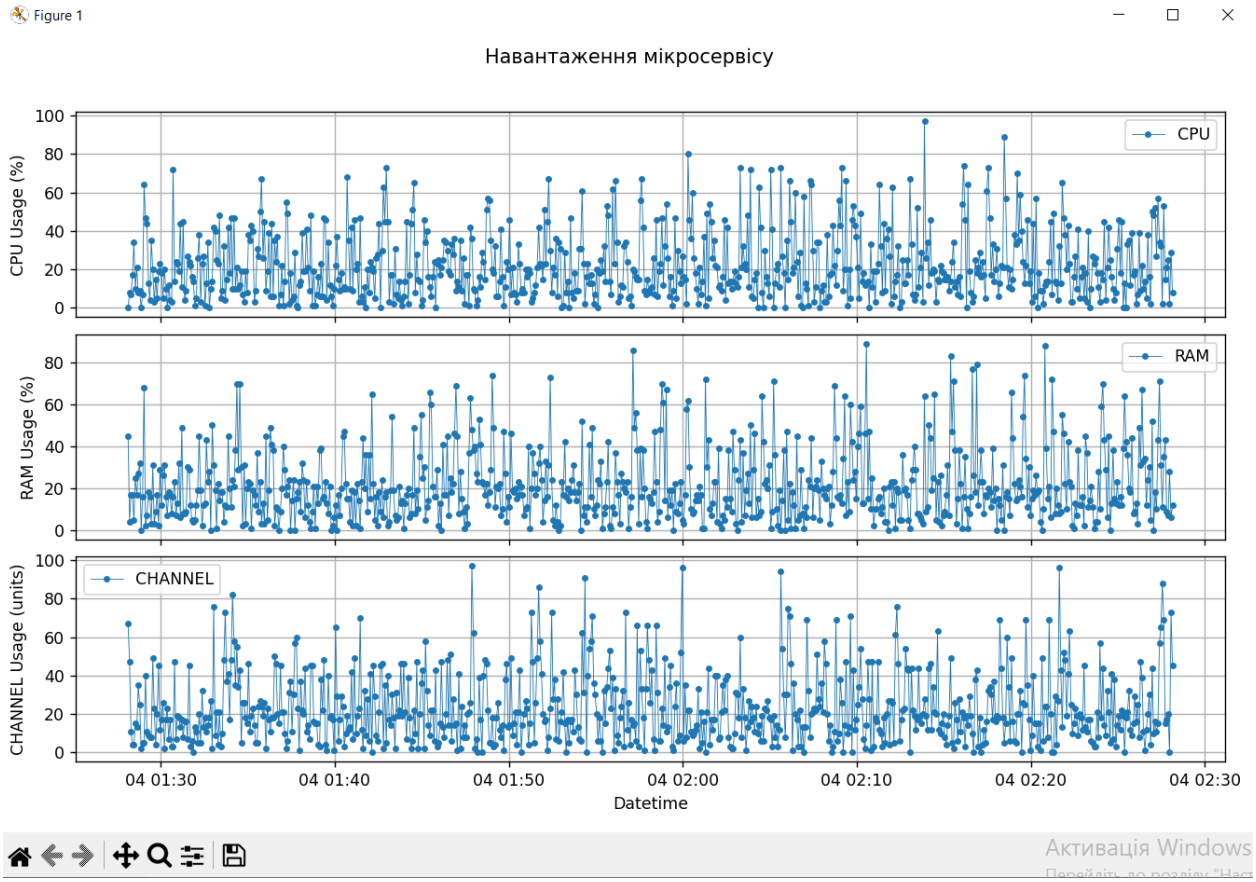


Рисунок 4.8 – Попередній перегляд вибраного датасету у вигляді графіку

Для генерування часових рядів потрібно задати потрібні налаштування (рис. 4.9).

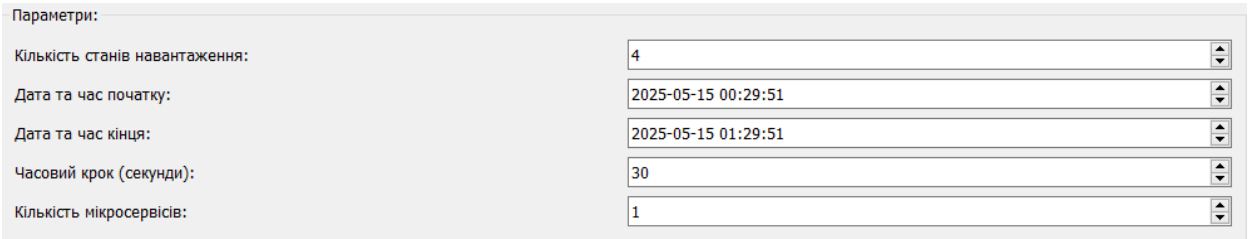


Рисунок 4.9 – Параметри для генерування часових рядів

Після натискання кнопки «Згенерувати часовий ряд», у разі успіху, користувач отримує повідомлення про те, що часовий ряд було згенеровано успішно (рис. 4.10).

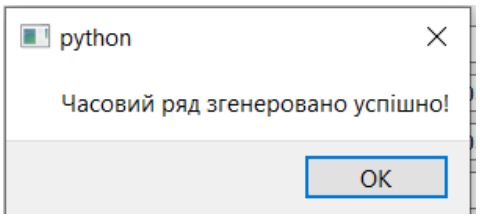


Рисунок 4.10 – Повідомлення про успішне генерування часового ряду

При натисканні кнопки «Зберегти» в полі збереження часового ряду в базу даних, при успішному збереженні, якщо назва групи мікросервісів не зустрічається в БД, буде виведено повідомлення, що часовий ряд було успішно збережено до бази даних (рис. 4.11).

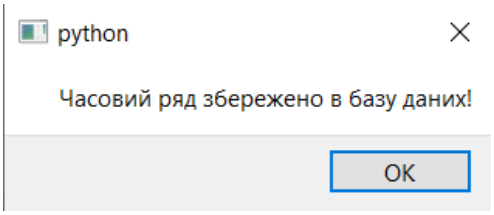


Рисунок 4.11 – Повідомлення про успішне збереження часового ряду до БД

Після генерування часового ряду, та, відповідно, після налаштування матриці переходів, користувач може переглянути ці матриці для кожного параметру навантаження у вигляді таблиць (рис. 4.12).

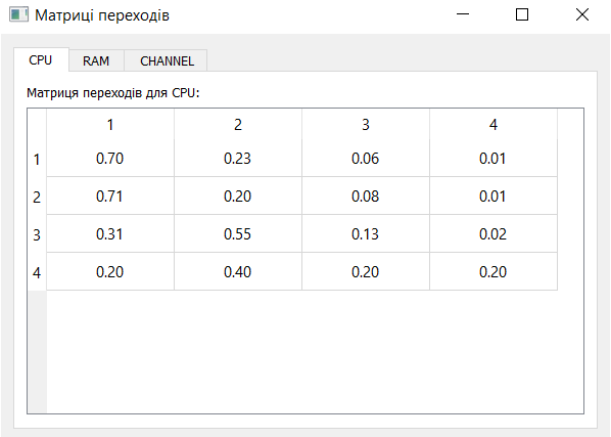


Рисунок 4.12 – Вікно відображення матриці переходів

Вікно відображення матриці переходів містить окремі таблиці для кожного з параметрів. Так, користувач може також переглянути матрицю переходів для використаних ресурсів ОЗП (рис. 4.13).

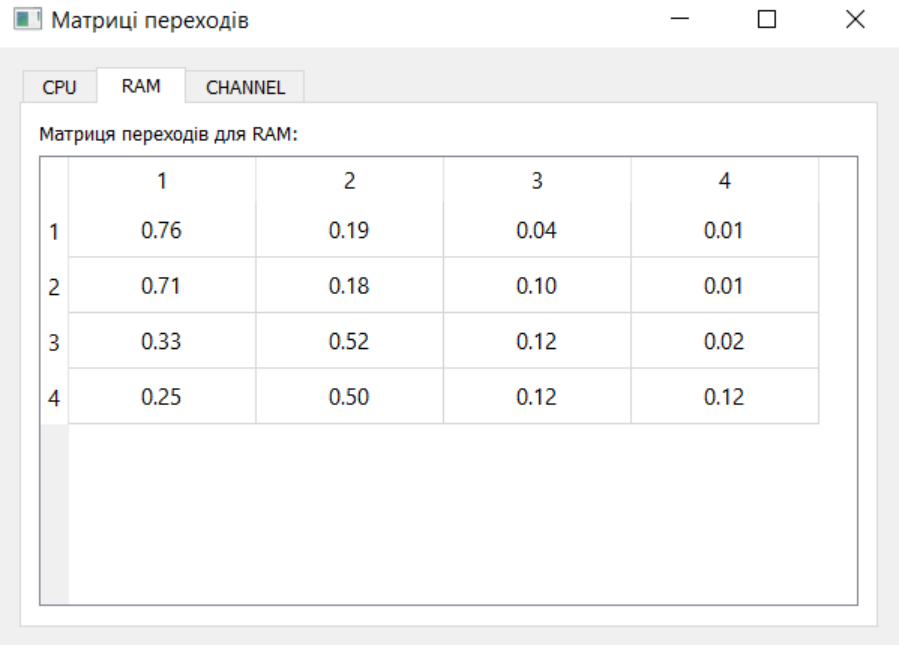


Рисунок 4.13 – Матриця переходів для використаної оперативної пам’яті

А також, матрицю переходів для використаного канального ресурсу (рис. 4.14).

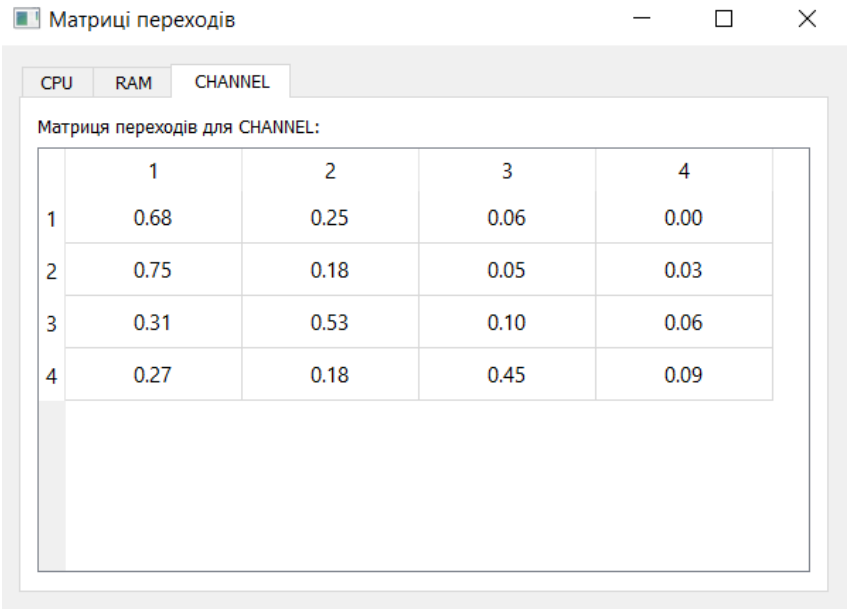


Рисунок 4.14 – Матриця переходів для використаного канального ресурсу

Якщо порівняти три матриці переходів для трьох параметрів навантаження, то можна побачити, що вони містять різні значення, що означає, що для кожного параметру будується своя матриця переходів, за якою. генеруються часові ряди.

Після генерування часового ряду користувач може обрати номер мікросервісу, для якого потрібно побудувати графік (рис. 4.15).

Рисунок 4.15 – Вибір мікросервісу для побудови графіку

При натисканні на кнопку «Побудувати графік» користувач отримує графік навантаження обраного мікросервісу (рис. 4.16).

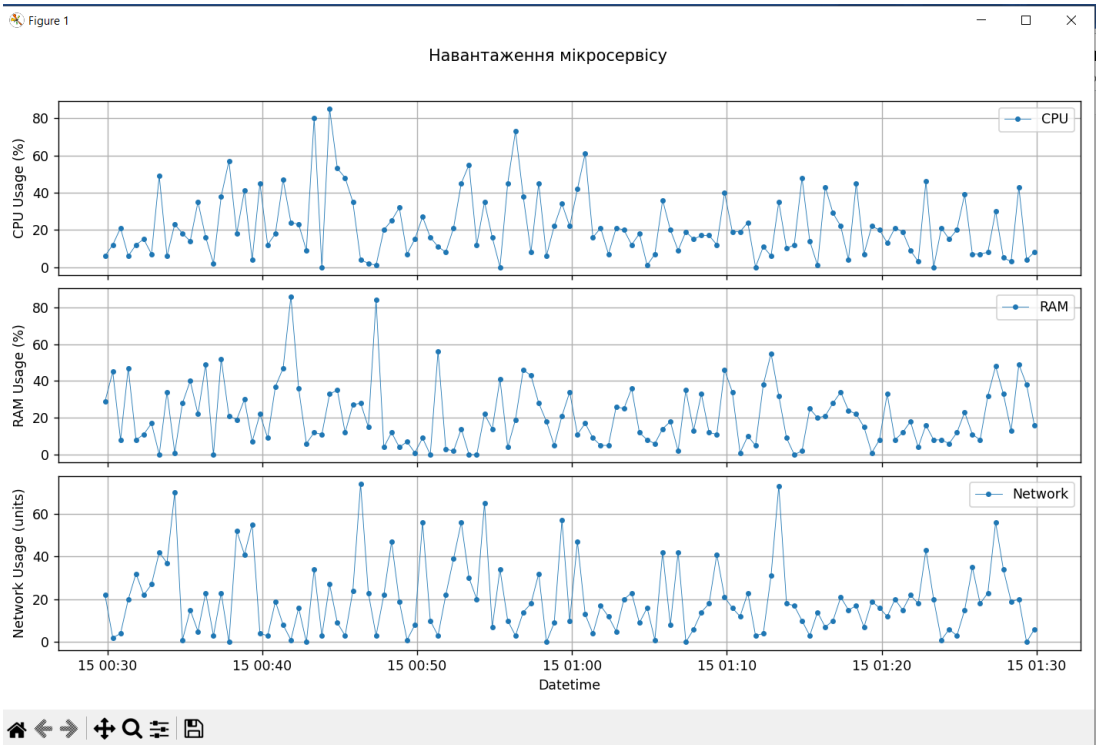


Рисунок 4.16 – Графік навантаження мікросервісу

Якщо порівняти графіки різних мікросервісів, згенерованих разом, можна побачити, що їхні графіки відрізняються (рис. 4.17).

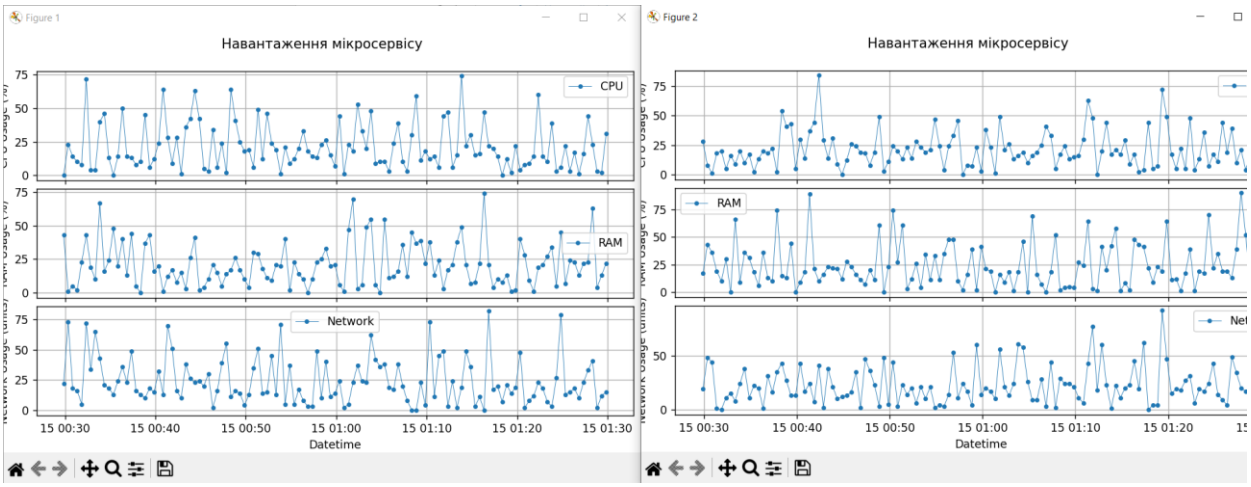


Рисунок 4.17 – Порівняння графіків мікросервісів

Окрім збереження згенерованих часовиї рядів до бази даних, користувач також може зберігати їх у вигляді .csv файлів. Для цього йому потрібно натиснути кнопку «Файл» і у випадаючому меню натиснути на кнопку «Зберегти датасет». Після цього відкриється вікно, схоже до вікна додавання датасету, в якому користувачеві потрібно вибрати папку, до якої збережуться згенеровані датасети (рис. 4.18).

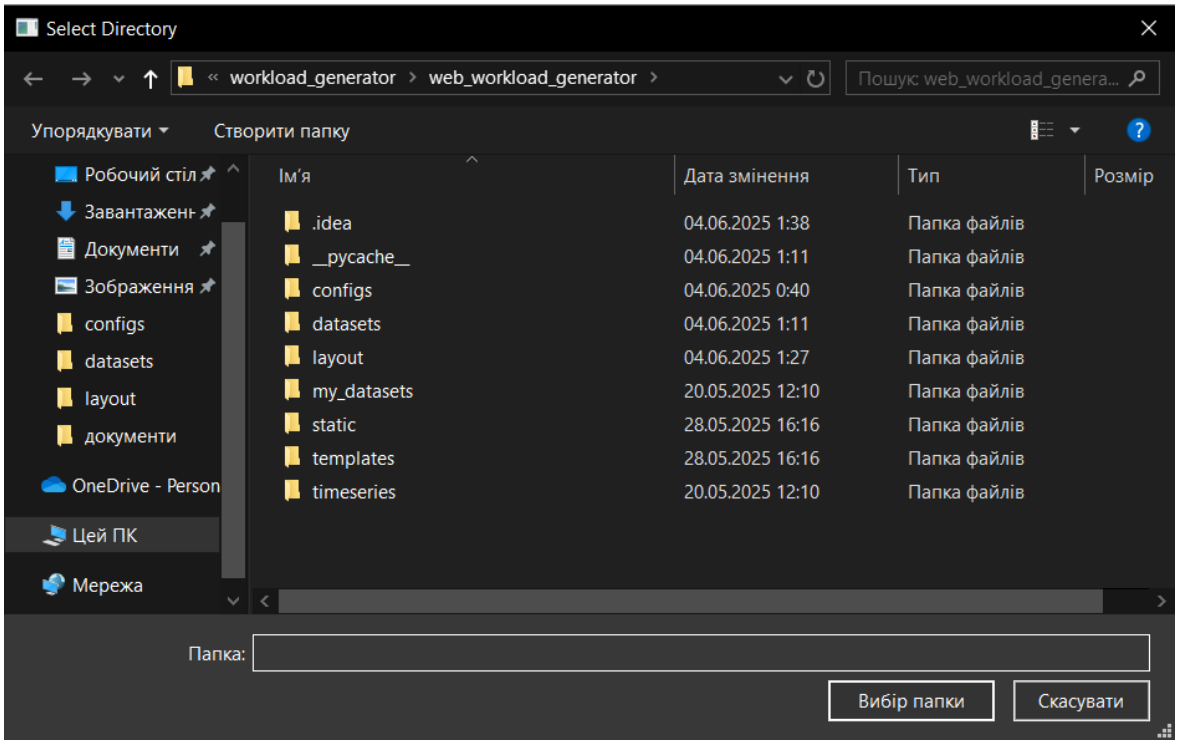


Рисунок 4.18 – Вікно вибору папки для збереження часових рядів

Для того, щоб дані в полях вводу зберігалися в тому стані, в якому вони були при закритті вікна програми, було створено файл ui_config.json, який має структуру, показану на рисунку 4.19.

Рисунок 4.19 – Структура файлу ui_config.json

Висновки до розділу 4

У четвертому розділі було розроблено графічний інтерфейс користувача для використання алгоритму генерування часових рядів. Виділено такі основні сценарії роботи користувача з розробленим GUI системи:

- вибирання типу налаштування матриці переходів;
- обирання файлів;
- налаштування параметрів генерування часових рядів;
- відображення матриці переходів;
- відображення графіків часових рядів;
- відображення підказок при наведенні на поля вводу.

ВИСНОВКИ

У даній дипломній роботі реалізовано алгоритм генерування часових рядів, які імітують навантаження в системах мікросервісів.

Вирішено задачу, яка полягає в наданні можливості тестувати алгоритми автоматичного розподілу ресурсів, за допомогою згенерованих часових рядів, що відображають використання серверних ресурсів мікросервісами.

Проаналізовано існуючі рішення для генерування часових рядів, обрано модель ланцюгів Маркова для реалізації алгоритму. Обґрунтовано вибір саме цього методу через можливість генерування даних без вхідних датасетів, генерування часових рядів, у яких наступні значення не залежать від послідовності попередніх та отримання різних часових рядів при однакових налаштуваннях.

Створено програмний продукт, який дозволяє використовувати розроблений алгоритм. Система дозволяє налаштовувати параметри та матриці переходів для генерування часових рядів, візуалізувати згенеровані часові ряди та матриці переходів. Програма також дозволяє зберігати згенеровані дані до бази даних та у вигляді файлів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. О.А. Дмитренко, "Мікросервісна архітектура та її задачі на прикладах з реального життя", Зібрання тез до конференції “XVII Міжнародна науково-технічна конференція "Перспективи телекомунікацій 2023”, 18-21 квітня, 2023. Київ. УДК 004.75. с. 239-242

2. Мікросервісна архітектура URL: https://medium.com/@IvanZmerzlyi/microservices-architecture-461687045b3d (дата звернення 08.05.2025)

3. О. Дмитренко і М. Скулиш, «Групування мікропослуг для використання неповно залучених ресурсів», Сучасні Інформаційні Технології, вип. 2024, вип. 1(3), с. 78–85, Бер 2025

4. М. А. Скулиш і О. А. Дмитренко, «Метод організації мікросервісів на серверних групах Kubernetes», Sci. Notes Taurida Natl. VI Vernadsky Univ. Ser. Tech. Sci., вип. 1, вип. 5, с. 291–297, Груд 2024, doi: 10.32782/2663-5941/2024.5.1/41.

5. What is Cloud Infrastructure? | Definition from TechTarget URL: https://www.techtarget.com/searchcloudcomputing/definition/cloud-infrastructure (дата звернення 09.05.2025)

6. What are ARIMA Models? | IBM URL: https://www.ibm.com/think/topics/arima-model (дата звернення 09.05.2025)

7. Kontopoulou, Vaia & Panagopoulos, Athanasios & Kakkos, Ioannis & Matsopoulos, George. (2023). A Review of ARIMA vs. Machine Learning Approaches for Time Series Forecasting in Data Driven Networks. Future Internet. 15. 255. 10.3390/fi15080255. URL: https://www.researchgate.net/publication/372771636 (дата звернення 09.05.2025)

8. What is Random Forest? | IBM URL: https://www.ibm.com/think/topics/random-forest (дата звернення 09.05.2025)

9. Biau, Gérard & Scornet, Erwan. (2015). A Random Forest Guided Tour. TEST. 25. 10.1007/s11749-016-0481-7. URL: https://www.researchgate.net/publication/284219299 (дата звернення 09.05.2025)

10. What is a Recurrent Neural Network (RNN)? URL: https://www.ibm.com/think/topics/recurrent-neural-networks (дата звернення 09.05.2025)

11. Lipton, Zachary. (2015). A Critical Review of Recurrent Neural Networks for Sequence Learning. URL: https://www.researchgate.net/publication/277603865 (дата звернення 09.05.2025)

12. What is a Markov Chain? URL: https://dida.do/what-is-a-markov-chain (дата звернення 09.05.2025)

13. Yan, Yuzhe. (2025). The Principle of the Markov Chain Prediction Model and Its Application in Stock Price Forecasting. Highlights in Science, Engineering and Technology. 140. 24-31. 10.54097/v3fpeq16. URL: https://www.researchgate.net/publication/392029208 (дата звернення 09.05.2025)

14. Розділ 2. Основи UML URL: https://docs.kde.org/stable5/uk/umbrello/umbrello/uml-basics.html (дата звернення 10.05.2025).

15. Інструкція, як будувати UML-діаграми - DOU URL: https://dou.ua/forums/topic/40575/ii-zir/ (дата звернення 10.05.2025).

16. Лекція 15. Нотація DFD URL: https://elib.lntu.edu.ua/sites/default/files/elib_upload/DFD (дата звернення 10.05.2025).

17. Лекція 6. Нотація IDEF0 URL: https://elib.lntu.edu.ua/sites/default/files/elib_upload/idef0 (дата звернення 22.05.2025).

18. PyQt – Python Wiki URL: https://wiki.python.org/moin/PyQt (дата звернення 11.05.2025).

ДОДАТОК А Лістинг розробленої системи

workload_generator.py

```
import json
import numpy as np
import csv
from datetime import datetime
import plot
import db


class MarkovChain:
    def __init__(self, states=4):
        self.__states = states
        self.__transition_matrix = {
            "CPU": np.zeros((states, states)),
            "RAM": np.zeros((states, states)),
            "CHANNEL": np.zeros((states, states)),
        }
        self.__timeseries = [{
            "Date": [],
            "Time": [],
            "CPU": [],
            "RAM": [],
            "CHANNEL": [],
        }]
        self.__current_states = {'CPU': 0, 'RAM': 0, 'CHANNEL': 0}
        self.__is_configured = False
        self.__dataset_id = 0
        self.__config_id = 0

    @property
    def transition_matrix(self):
        return self.__transition_matrix

    def set_config(self):
        try:
            self.load_config()
        except (FileNotFoundError, json.decoder.JSONDecodeError, ValueError) as e:
            self.configure()

    def first_state(self, key):
        states_probabilities_sum = [0 for _ in range(self.__states)]
        for i in range(self.__states):
            for j in range(self.__states):
                states_probabilities_sum[j] += self.__transition_matrix[key][i][j]
        return states_probabilities_sum.index(max(states_probabilities_sum))

    @staticmethod
    def prepare_count_matrix(transition_count_matrix):
        for key in transition_count_matrix:
            for i in range(len(transition_count_matrix[key])):
                for j in range(len(transition_count_matrix[key][i])):
                    if transition_count_matrix[key][i][j] == 0:
                        transition_count_matrix[key][i][j] = 1

    def get_state_from_value(self, value):
```

```
        for i in range(1, self.__states + 1):
            if value <= i * (100 / self.__states):
                return i - 1

    def count_transitions(self, dataset):
        transition_count = {
            'CPU': [[0 for _ in range(self.__states)] for _ in range(self.__states)],
            'RAM': [[0 for _ in range(self.__states)] for _ in range(self.__states)],
            'CHANNEL': [[0 for _ in range(self.__states)] for _ in range(self.__states)],
        }

        current_states = {'CPU': self.get_state_from_value(dataset['CPU'][0]),
                          'RAM': self.get_state_from_value(dataset['RAM'][0]),
                          'CHANNEL': self.get_state_from_value(dataset['CHANNEL'][0])}

        for i in range(1, len(dataset['Date'])):
            next_cpu_state = self.get_state_from_value(dataset['CPU'][i])
            transition_count['CPU'][current_states['CPU']][next_cpu_state] += 1
            current_states['CPU'] = next_cpu_state

            next_ram_state = self.get_state_from_value(dataset['RAM'][i])
            transition_count['RAM'][current_states['RAM']][next_ram_state] += 1
            current_states['RAM'] = next_ram_state

            next_network_state = self.get_state_from_value(dataset['CHANNEL'][i])
            transition_count['CHANNEL'][current_states['CHANNEL']][next_network_state] +=
1
            current_states['CHANNEL'] = next_network_state

        return transition_count

    def create_transition_matrix(self, transition_count):
        for key in transition_count:
            for i in range(len(transition_count[key])):
                self.prepare_count_matrix(transition_count)
                state_count_sum = sum(transition_count[key][i])
                multiplier = 1 / state_count_sum
                for j in range(len(transition_count[key][i]) - 1):
                    self.__transition_matrix[key][i][j] = multiplier *
transition_count[key][i][j]
                self.__transition_matrix[key][i][-1] = 1 -
sum(self.__transition_matrix[key][i])

    def configure(self, path="timeseries.csv"):
        dataset_id = db.get_dataset_id_by_name(path)
        if not dataset_id:
            raise FileNotFoundError("Такого датасету не існує")
        transition_matrix = db.get_config_for_dataset(dataset_id, self.__states)
        if transition_matrix:
            self.__transition_matrix = transition_matrix
            self.__is_configured = True
            return
        self.__dataset_id = dataset_id
        dataset = db.get_dataset_by_id(dataset_id)
        transition_count = self.count_transitions(dataset)

        self.create_transition_matrix(transition_count)

        self.__is_configured = True
```

```
        self.save_config(path)

    @staticmethod
    def prepare_config_to_json(transition_matrix: dict):
        json_transition_matrix = transition_matrix.copy()
        for key in transition_matrix:
            json_transition_matrix[key] = transition_matrix[key].tolist()
        return json_transition_matrix

    def set_states(self, states):
        if not isinstance(states, int):
            raise TypeError("states must be an integer")
        if states < 1 or states > 20:
            raise ValueError("states must be between 1 and 20")
        self.__states = states
        self.__transition_matrix = {
            "CPU": np.zeros((states, states)),
            "RAM": np.zeros((states, states)),
            "CHANNEL": np.zeros((states, states)),
        }

    def save_config(self, dataset_path):
        db.save_auto_config(self.__transition_matrix,
db.get_dataset_id_by_name(dataset_path))

    def load_config(self, path="config.json"):
        transition_matrix = db.get_config_by_name(path)
        if len(transition_matrix['CPU']) != self.__states:
            raise ValueError(f"Кількість станів навантаження в конфігураційому файлі
({len(transition_matrix['CPU'])})"
                             f" не збігається з заданою кількістю станів
({self.__states})")
        self.__transition_matrix = transition_matrix
        self.__is_configured = True

    def __next_value(self, current_state, component):
        next_state = np.random.choice(self.__states,
p=self.__transition_matrix[component][current_state])
        self.__current_states[component] = next_state
        bottom_state_value = int(100 / self.__states) * next_state
        top_state_value = int(100 / self.__states) * (next_state + 1)
        return np.random.randint(bottom_state_value, top_state_value)

    def __generate_timeseries(self, from_timestamp, to_timestamp, step=1000):
        timeseries = {"Date": [],
                      "Time": [],
                      "CPU": [],
                      "RAM": [],
                      "CHANNEL": []}
        self.__current_states = {'CPU': self.first_state('CPU'),
                                 'RAM': self.first_state('RAM'),
                                 'CHANNEL': self.first_state('CHANNEL')}

        for timestamp in range(from_timestamp, to_timestamp + 1, step):
            timeseries['Date'].append(datetime.fromtimestamp(timestamp).date())
            timeseries['Time'].append(datetime.fromtimestamp(timestamp).time())

            timeseries['CPU'].append(self.__next_value(self.__current_states['CPU'],
'CPU'))
```

```
            timeseries['RAM'].append(self.__next_value(self.__current_states['RAM'],
'RAM'))

timeseries['CHANNEL'].append(self.__next_value(self.__current_states['CHANNEL'],
'CHANNEL'))

        return timeseries

    def generate(self, from_timestamp, to_timestamp, step=1000, count=1):
        self.__timeseries = []
        if not self.__is_configured:
            raise ValueError("Потрібно налаштувати матрицю переходів")
        for i in range(count):
            self.__timeseries.append(self.__generate_timeseries(from_timestamp,
to_timestamp, step))


        return self.__timeseries.copy()

    def save_to_db(self, group_name):
        names = db.get_datasets_names()
        group_names = []
        for name in names:
            last_slash_index = name.rfind("/")
            if last_slash_index == -1:
                group_names.append(name)
            else:
                group_names.append(name[:last_slash_index])
        if group_name in group_names:
            raise ValueError('Група мікросервісів з такою назвою вже існує')
        db.save_timeseries(self.__timeseries, group_name)

    def save_file(self, directory='timeseries'):
        for num, timeseries in enumerate(self.__timeseries):
            with open(f'{directory}/{num + 1}.csv', 'w', newline='') as f:
                writer = csv.writer(f)
                writer.writerow(['Date', 'Time', 'CPU', 'RAM', 'CHANNEL'])
                for i in range(0, len(timeseries['Date'])):
                    writer.writerow([timeseries['Date'][i], timeseries['Time'][i],
timeseries['CPU'][i], timeseries['RAM'][i], timeseries['CHANNEL'][i]])

    def show_plot(self, index):
        if index < 0 or index >= len(self.__timeseries):
            raise IndexError("Index out of range")
        dataset = self.__timeseries[index]
        plot.plot_dataset(dataset)
```

UI.py

```
import os
import PyQt5
from PyQt5.QtGui import QCloseEvent
from PyQt5.uic import loadUi
from PyQt5.QtWidgets import QMainWindow, QFileDialog, QMessageBox, QWidget, QVBoxLayout,
QTabWidget, \
    QTableWidget, QTableWidgetItem, QLabel
from PyQt5 import QtCore
import db
import workload_generator
```

```
import json
from datetime import datetime
import load_data
import plot



class TransitionMatrixWindow(QWidget):
    def __init__(self, matrix_data):
        super().__init__()
        self.setWindowTitle("Матриці переходів")
        self.resize(600, 400)

        layout = QVBoxLayout()
        tabs = QTabWidget()

        for label, matrix in matrix_data.items():
            tab = QWidget()
            tab_layout = QVBoxLayout()

            table = QTableWidget()
            num_rows = len(matrix)
            num_cols = len(matrix[0]) if matrix else 0

            table.setRowCount(num_rows)
            table.setColumnCount(num_cols)

            for i in range(num_rows):
                for j in range(num_cols):
                    item = QTableWidgetItem(f"{matrix[i][j]:.2f}")
                    item.setTextAlignment(PyQt5.QtCore.Qt.AlignCenter)
                    table.setItem(i, j, item)

            tab_layout.addWidget(QLabel(f"Матриця переходів для {label}:"))
            tab_layout.addWidget(table)
            tab.setLayout(tab_layout)
            tabs.addTab(tab, label)

        layout.addWidget(tabs)
        self.setLayout(layout)


class MainWindow(QMainWindow):

    def __init__(self):
        super(MainWindow, self).__init__()
        self.__workload_generator = workload_generator.MarkovChain()
        self.__dataset_path = 'datasets/workload_dataset/workload.csv'
        self.__config_path = 'middle_config.json'
        self.__configure_mode = 'config'
        self.__transition_matrix_window = None
        self.setup()

    def setup(self):
        loadUi("layout/MainWindow.ui", self)
        self.chooseDatasetRadioButton.toggled.connect(lambda:
self.__set_configure_mode("dataset"))
        self.addDatasetButton.clicked.connect(lambda : self.debug(self.choose_dataset))
        self.chooseConfigRadioButton.toggled.connect(lambda:
```

```
self.__set_configure_mode("config"))
        self.addConfigButton.clicked.connect(lambda : self.debug(self.choose_config))
        self.generateButton.clicked.connect(lambda: self.debug(self.generate))
        (self.plotButton.clicked.connect(
            lambda:
self.__workload_generator.show_plot(self.microserviceNumberComboBox.currentIndex()))
        )
        self.startDateTimeEdit.setDateTime(QtCore.QDateTime.currentDateTime())
        self.endDateTimeEdit.setDateTime(QtCore.QDateTime.currentDateTime().addSecs(3600))
        self.startDateTimeEdit.setDisplayFormat("yyyy-MM-dd HH:mm:ss")
        self.endDateTimeEdit.setDisplayFormat("yyyy-MM-dd HH:mm:ss")
        self.actionSave.triggered.connect(lambda: self.debug(self.save_dataset))
        self.saveTimeseriesToDBButton.clicked.connect(lambda:
self.debug(self.save_timeseries_to_db))
        self.plotDatasetButton.clicked.connect(lambda: self.debug(self.plot_dataset))
        self.chooseDatasetComboBox.addItems(db.get_datasets_names())
        self.chooseConfigComboBox.addItems(db.get_manual_configs_names())
        self.showTransitionMatrixButton.clicked.connect(lambda:
self.debug(self.show_transition_matrix))
        self.load_config()

    @staticmethod
    def debug(function, *args):
        try:
            function(*args)
        except Exception as ex:
            msg = QMessageBox()
            msg.setText(str(ex))
            msg.exec_()

    def __set_dataset_path(self, dataset_path):
        self.__dataset_path = dataset_path

    def __set_configure_mode(self, mode: str):
        self.__configure_mode = mode

    def plot_dataset(self):
        if self.chooseDatasetComboBox.currentText():

plot.plot_dataset(db.get_dataset_by_name(self.chooseDatasetComboBox.currentText()))

    def load_config(self):
        try:
            with open('ui_config.json', 'r', encoding='utf-8') as f:
                config = json.load(f)
                if 'configMode' in config:
                    if config['configMode'] in ('dataset', 'config'):
                        self.__configure_mode = config['configMode']
                    if config['configMode'] == 'config':
                        self.chooseConfigRadioButton.setChecked(True)
                        self.chooseDatasetRadioButton.setChecked(False)
                    elif config['configMode'] == 'dataset':
                        self.chooseConfigRadioButton.setChecked(False)
                        self.chooseDatasetRadioButton.setChecked(True)
                if 'statesSpinBox' in config:
                    self.statesSpinBox.setValue(int(config['statesSpinBox']))
                if 'chooseConfigComboBox' in config:

self.chooseConfigComboBox.setCurrentText(config['chooseConfigComboBox'])
```

```
                if 'chooseDatasetComboBox' in config:

self.chooseDatasetComboBox.setCurrentText(config['chooseDatasetComboBox'])
                if 'stepSpinBox' in config:
                    self.stepSpinBox.setValue(int(config['stepSpinBox']))
                if 'microservicesCountSpinBox' in config:

self.microservicesCountSpinBox.setValue(int(config['microservicesCountSpinBox']))
        except Exception as ex:
            pass

    def save_config(self):
        config = {
            'configMode': 'config' if self.__configure_mode == 'config' else 'dataset',
            "statesSpinBox": self.statesSpinBox.value(),
            "chooseConfigComboBox": self.chooseConfigComboBox.currentText(),
            "chooseDatasetComboBox": self.chooseDatasetComboBox.currentText(),
            "stepSpinBox": self.stepSpinBox.value(),
            "microservicesCountSpinBox": self.microservicesCountSpinBox.value(),
        }
        with open('ui_config.json', 'w', encoding='utf-8') as f:
            json.dump(config, f, ensure_ascii=False)

    def config_from_dataset(self):
        self.__workload_generator.configure(self.chooseDatasetComboBox.currentText())

    def generate(self):
        self.__workload_generator.set_states(self.statesSpinBox.value())
        if self.__configure_mode == 'config':
            self.__workload_generator.load_config(self.chooseConfigComboBox.currentText())
        else:
            self.config_from_dataset()

        from_timestamp =
int(datetime.strptime(self.startDateTimeEdit.dateTime().toString(self.startDateTimeEdit.di
splayFormat()), "%Y-%m-%d %H:%M:%S").timestamp())
        to_timestamp =
int(datetime.strptime(self.endDateTimeEdit.dateTime().toString(self.endDateTimeEdit.displa
yFormat()), "%Y-%m-%d %H:%M:%S").timestamp())
        self.__workload_generator.generate(from_timestamp, to_timestamp,
self.stepSpinBox.value(), self.microservicesCountSpinBox.value())

        msg = QMessageBox()
        msg.setText("Часовий ряд згенеровано успішно!")
        msg.exec_()

        self.plotButton.setEnabled(True)
        self.saveTimeseriesToDBButton.setEnabled(True)
        self.showTransitionMatrixButton.setEnabled(True)
        self.microserviceNumberComboBox.clear()
        self.microserviceNumberComboBox.addItems(list(str(i) for i in range(1,
self.microservicesCountSpinBox.value() + 1)))

    def choose_dataset(self):
        fname = QFileDialog(self).getOpenFileName(self, 'Open file',

os.path.dirname(os.path.abspath(__file__)), "CSV files (*.csv)")
        if fname[0]:
            dataset = load_data.load_dataset(fname[0])
```

```
            db.save_dataset(dataset, fname[0])
            self.chooseDatasetComboBox.addItem(fname[0])
            self.chooseDatasetComboBox.setCurrentText(fname[0])
            self.__dataset_path = fname[0]

    def show_transition_matrix(self):
        transition_matrix = self.__workload_generator.transition_matrix
        self.__transition_matrix_window = TransitionMatrixWindow(transition_matrix)
        self.__transition_matrix_window.show()

    def choose_config(self):
        fname = QFileDialog(self).getOpenFileName(self, 'Open file',

os.path.dirname(os.path.abspath(__file__)), "JSON files (*.json)")
        if fname[0]:
            config = load_data.load_config(fname[0])
            db.save_manual_config(config, fname[0])
            self.chooseConfigComboBox.addItem(fname[0])
            self.chooseConfigComboBox.setCurrentText(fname[0])
            self.__config_path = fname[0]

    def save_timeseries_to_db(self):
        name = self.microservicesGroupNameLineEdit.text()
        if not name:
            raise ValueError("Вкажіть назву групи мікросервісів")

        self.__workload_generator.save_to_db(name)

        self.chooseDatasetComboBox.addItems(
            [f'{name}/{i + 1}.csv' for i in
range(self.microservicesCountSpinBox.value())])

        msg = QMessageBox()
        msg.setText("Часовий ряд збережено в базу даних!")
        msg.exec_()

    def save_dataset(self):
        options = QFileDialog.ShowDirsOnly
        fname = QFileDialog.getExistingDirectory(self, "Select Directory", "",
options=options)
        if not fname:
            return
        self.__workload_generator.save_file(fname)

    def closeEvent(self, event: QCloseEvent):
        self.save_config()
        event.accept()
```

plot.py

```
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime


matplotlib.use("TkAgg")
```

```
def plot_dataset(dataset):
    df = pd.DataFrame(dataset)

    df['Datetime'] = [datetime.combine(d, t) for d, t in zip(df["Date"], df["Time"])]
    fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(12, 10), sharex=True)
    fig.suptitle('Навантаження мікросервісу')

    axes[0].plot(df['Datetime'], df['CPU'], label='CPU', marker='o', linestyle='-',
linewidth=0.5, markersize=3)
    axes[0].set_ylabel('CPU Usage (%)')
    axes[0].legend()
    axes[0].grid(True)

    axes[1].plot(df['Datetime'], df['RAM'], label='RAM', marker='o', linestyle='-',
linewidth=0.5, markersize=3)
    axes[1].set_ylabel('RAM Usage (%)')
    axes[1].legend()
    axes[1].grid(True)

    axes[2].plot(df['Datetime'], df['CHANNEL'], label='CHANNEL', marker='o', linestyle='-
', linewidth=0.5, markersize=3)
    axes[2].set_ylabel('CHANNEL Usage (units)')
    axes[2].set_xlabel('Datetime')
    axes[2].legend()
    axes[2].grid(True)

    plt.tight_layout(rect=(0.0, 0.03, 1.0, 0.95))
    plt.show()
```

load_data.py

```
import csv
from datetime import datetime
import os
import json


def load_dataset(path):
    dataset = {
            "Date": [],
            "Time": [],
            "CPU": [],
            "RAM": [],
            "CHANNEL": [],
        }
    with open(path, "r", encoding='utf-8') as f:
        reader = csv.reader(f)
        first_row = next(reader)
        if len(first_row) != 5 or first_row[0] != "Date" or first_row[1] != "Time" or
first_row[2] != "CPU" or first_row[3] != "RAM" or first_row[4] != "CHANNEL":
            raise ValueError("Датасет повинен містити параметри: Date, Time, CPU, RAM,
CHANNEL")
        for i, row in enumerate(reader):
            try:
                date_time = datetime.strptime(row[0] + ' ' + row[1], '%Y-%m-%d %H:%M:%S')
            except Exception:
                raise ValueError(f"Рядок {i + 2}: Неправильний формат дати чи
часу.\nКоректний формат: %Y-%m-%d %H:%M:%S")
            dataset["Date"].append(date_time.date())
```

```
            dataset["Time"].append(date_time.time())
            try:
                float(row[2])
                float(row[3])
                float(row[4])
            except Exception:
                raise ValueError(f"Рядок {i + 2}: Дані навантаження повинні бути у
числовому вигляді")
            if float(row[2]) < 0 or float(row[2]) > 100 or \
                float(row[3]) < 0 or float(row[3]) > 100 or \
                float(row[4]) < 0 or float(row[4]) > 100:
                raise ValueError(f"Рядок {i + 2}: Значення навантаження мають бути в межах
0-100")
            dataset["CPU"].append(float(row[2]))
            dataset["RAM"].append(float(row[3]))
            dataset["CHANNEL"].append(float(row[4]))

    return dataset


def check_config(config):
    if 'States' not in config:
        raise ValueError("Конфігураційний файл повинен містити параметр 'States'(кількість
станів навантаження)")
    if 'MarkovChain' not in config:
        raise ValueError("Конфігураційний файл повинен містити параметр
'MarkovChain'(матриця переходів)")
    states = config['States']
    transition_matrix = config['MarkovChain']
    if len(transition_matrix["CPU"]) != states or \
            len(transition_matrix["RAM"]) != states or \
            len(transition_matrix["CHANNEL"]) != states:
        raise ValueError("Неправильна конфігурація. Розмір матриці має бути рівним
кількості станів")
    for i in range(len(transition_matrix["CPU"])):
        if len(transition_matrix["CPU"][i]) != states or \
                len(transition_matrix["RAM"][i]) != states or \
                len(transition_matrix["CHANNEL"][i]) != states:
            raise ValueError("Неправильна конфігурація. Матриці мають бути квадратними.")
    for key in transition_matrix:
        for state_probabilities in transition_matrix[key]:
            if sum(state_probabilities) != 1:
                raise ValueError('Неправильна конфігурація. Сума ймовірностей кожного
рядка повинна бути рівною 1.')
    for key in transition_matrix:
        for i in range(len(transition_matrix[key])):
            for j in range(len(transition_matrix[key])):
                if transition_matrix[key][i][j] <= 0 or transition_matrix[key][i][j] >= 1:
                    raise ValueError("Значення ймовірностей повинні бути в межах (0, 1)")

def load_config(path):
    if not os.path.isfile(path):
        raise FileNotFoundError(f"No {path} file")
    with open(path, "r", encoding='UTF-8') as f:
        json_data = json.load(f)
        json_data['AutoGenerated'] = 0
        try:
            check_config(json_data)
        except ValueError as e:
```

```
            raise ValueError(str(e))
        return json_data
```

db.py

```
import sqlite3
from datetime import datetime


def setup():
    connection = sqlite3.connect("workload.db")
    connection.execute("CREATE TABLE IF NOT EXISTS Datasets ("
                       "id INTEGER NOT NULL,"
                       "service_name TEXT NOT NULL,"
                       "metric_type TEXT CHECK ( metric_type IN ('CPU', 'RAM', 'CHANNEL')
),"
                       "date DATE NOT NULL,"
                       "time TIME NOT NULL,"
                       "value FLOAT NOT NULL,"
                       "PRIMARY KEY (date, time, service_name, metric_type)"
                       ")")

    connection.execute("CREATE TABLE IF NOT EXISTS Configurations ("
                       "id INTEGER NOT NULL,"
                       "states INTEGER NOT NULL,"
                       "is_auto_generated INTEGER NOT NULL DEFAULT 1,"
                       "name TEXT,"
                       "dataset_id INTEGER,"
                       "from_state INTEGER NOT NULL,"
                       "to_state INTEGER NOT NULL,"
                       "CPU FLOAT NOT NULL,"
                       "RAM FLOAT NOT NULL,"
                       "CHANNEL FLOAT NOT NULL,"
                       "FOREIGN KEY (dataset_id) REFERENCES Datasets(id) ON DELETE
CASCADE,"
                       "PRIMARY KEY (id, from_state, to_state)"
                       ")")

    connection.commit()


def insert_auto_config(row):
    connection = sqlite3.connect("workload.db")
    cursor = connection.cursor()
    cursor.execute(
        "INSERT INTO Configurations(id, states, is_auto_generated, dataset_id, from_state,
to_state, CPU, RAM, CHANNEL) "
        "VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?)", tuple(row))
    connection.commit()


def save_auto_config(transition_matrix, dataset_id):
    states = len(transition_matrix['CPU'])
    config = {
        "States": states,
        "is_auto_generated": 1,
        "dataset_id": dataset_id,
        "MarkovChain": transition_matrix
    }
```

```
    connection = sqlite3.connect("workload.db")
    cursor = connection.cursor()
    index = cursor.execute("SELECT id FROM Configurations ORDER BY id DESC LIMIT
1").fetchone()
    if not index:
        index = 1
    else:
        index = int(index[0]) + 1
    for i in range(config["States"]):
        for j in range(config["States"]):
            insert_auto_config((index, config["States"], config["is_auto_generated"],
config["dataset_id"], i, j,
                                transition_matrix['CPU'][i][j],
transition_matrix['RAM'][i][j],
                                transition_matrix['CHANNEL'][i][j]))


def insert_manual_config(row):
    connection = sqlite3.connect("workload.db")
    cursor = connection.cursor()
    cursor.execute(
        "INSERT INTO Configurations(id, states, is_auto_generated, name, from_state,
to_state, CPU, RAM, CHANNEL) "
        "VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?)", tuple(row))
    connection.commit()


def save_manual_config(config, name):
    config["is_auto_generated"] = 0
    config["name"] = name
    transition_matrix = config["MarkovChain"]
    connection = sqlite3.connect("workload.db")
    cursor = connection.cursor()
    index = cursor.execute("SELECT id FROM Configurations ORDER BY id DESC LIMIT
1").fetchone()
    if not index:
        index = 1
    else:
        index = int(index[0]) + 1
    for i in range(config["States"]):
        for j in range(config["States"]):
            insert_manual_config((index, config["States"], config["is_auto_generated"],
config["name"], i, j,
                                transition_matrix['CPU'][i][j],
transition_matrix['RAM'][i][j], transition_matrix['CHANNEL'][i][j]))


def insert_dataset(row):
    connection = sqlite3.connect("workload.db")
    cursor = connection.cursor()
    cursor.execute("INSERT INTO Datasets(id, service_name, metric_type, date, time, value)
VALUES(?, ?, ?, ?, ?, ?)", tuple(row))
    connection.commit()


def save_dataset(dataset: dict, dataset_name: str):
    connection = sqlite3.connect("workload.db")
    cursor = connection.cursor()
    index = cursor.execute("SELECT id FROM Datasets ORDER BY id DESC LIMIT 1").fetchone()
```

```
    if not index:
        index = 1
    else:
        index = int(index[0]) + 1

    metrics = list(dataset.keys())
    metrics.remove('Date')
    metrics.remove('Time')

    for metric in metrics:
        for i in range(len(dataset[metric])):
            row = (
                index,
                dataset_name,
                metric,
                str(dataset['Date'][i]),
                str(dataset['Time'][i]),
                dataset[metric][i]
            )
            insert_dataset(row)



def save_timeseries(timeseries, directory_name):
    for i in range(0, len(timeseries)):
        save_dataset(timeseries[i], directory_name + '/' + str(i + 1) + '.csv')


def get_datasets_names() -> list[str]:
    connection = sqlite3.connect("workload.db")
    cursor = connection.cursor()
    cursor.execute("SELECT DISTINCT service_name FROM Datasets ORDER BY id")
    datasets_names = cursor.fetchall()
    datasets_names = [record[0] for record in datasets_names]
    return datasets_names


def get_manual_configs_names():
    connection = sqlite3.connect("workload.db")
    cursor = connection.cursor()
    cursor.execute("SELECT DISTINCT name FROM Configurations WHERE is_auto_generated=0
ORDER BY id ")
    configs_names = cursor.fetchall()
    configs_names = [record[0] for record in configs_names]
    return configs_names


def get_dataset_by_id(dataset_id: int):
    connection = sqlite3.connect("workload.db")
    cursor = connection.cursor()
    cursor.execute("SELECT date, time, metric_type, value FROM Datasets WHERE id = ? ORDER
BY date, time", (dataset_id,))
    rows = cursor.fetchall()
    dataset = {"Date": [], "Time": [], "CPU": [], "RAM": [], "CHANNEL": []}
    for row in rows:
        date_time = datetime.strptime(row[0] + ' ' + row[1], '%Y-%m-%d %H:%M:%S')
        if not (len(dataset['Date']) != 0 and date_time.date() == dataset['Date'][-1] and
date_time.time() == dataset['Time'][-1]):
            dataset['Date'].append(date_time.date())
```

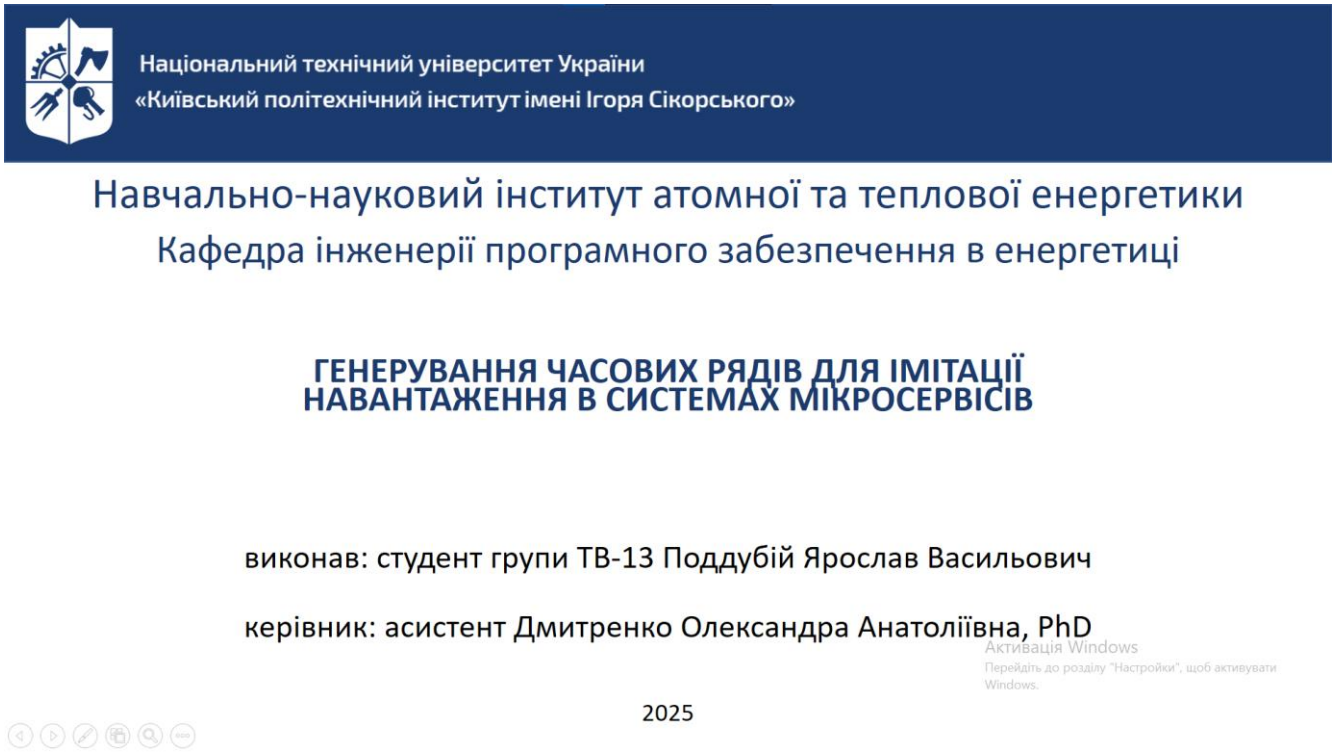
```
            dataset['Time'].append(date_time.time())
        dataset[row[2]].append(row[3])
    return dataset


def get_dataset_id_by_name(dataset_name: str):
    connection = sqlite3.connect("workload.db")
    cursor = connection.cursor()
    cursor.execute("SELECT id FROM Datasets WHERE service_name = ? LIMIT 1",
(dataset_name,))
    index = cursor.fetchone()
    if not index:
        return None
    index = int(index[0])
    return index


def get_dataset_by_name(dataset_name: str):
    index = get_dataset_id_by_name(dataset_name)
    if not index:
        return None
    dataset = get_dataset_by_id(index)
    return dataset


def get_config_by_id(index):
    connection = sqlite3.connect("workload.db")
    cursor = connection.cursor()
    cursor.execute("SELECT states, from_state, to_state, CPU, RAM, CHANNEL FROM
Configurations WHERE id = ?", (index,))
    rows = cursor.fetchall()
    if not rows:
        return None
    states = rows[0][0]
    transition_matrix = {
        "CPU": [[0 for _ in range(states)] for _ in range(states)],
        "RAM": [[0 for _ in range(states)] for _ in range(states)],
        "CHANNEL": [[0 for _ in range(states)] for _ in range(states)]}
    for row in rows:
        transition_matrix['CPU'][int(row[1])][int(row[2])] = float(row[3])
        transition_matrix['RAM'][int(row[1])][int(row[2])] = float(row[4])
        transition_matrix['CHANNEL'][int(row[1])][int(row[2])] = float(row[5])
    return transition_matrix


def get_config_by_name(config_name: str):
    connection = sqlite3.connect("workload.db")
    cursor = connection.cursor()
    cursor.execute("SELECT id FROM Configurations WHERE name = ? LIMIT 1", (config_name,))
    index = cursor.fetchone()
    if not index:
        return None
    index = int(index[0])
    config = get_config_by_id(index)
    return config


def get_config_for_dataset(dataset_id: int, states: int):
    connection = sqlite3.connect("workload.db")
```

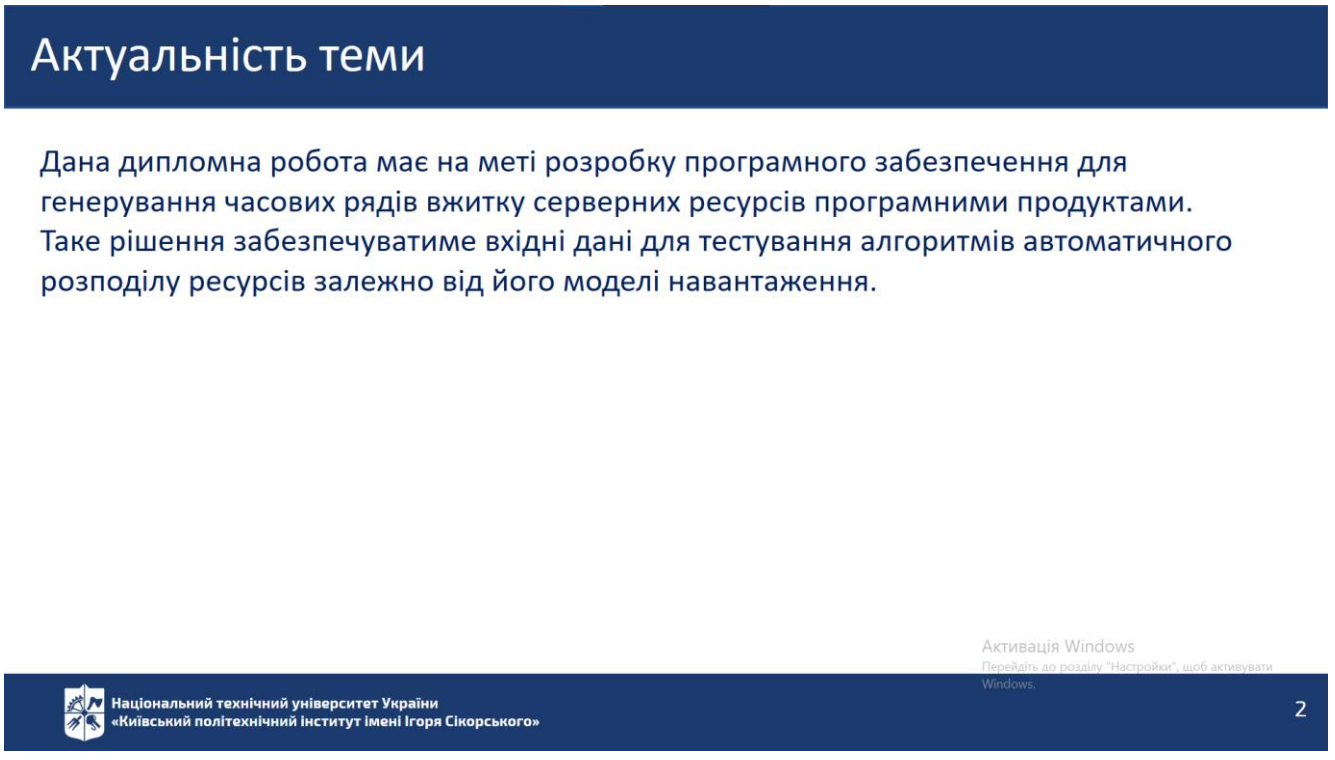
```
    cursor = connection.cursor()
    cursor.execute("SELECT id FROM Configurations WHERE dataset_id = ? AND states = ?
LIMIT 1", (dataset_id, states))
    index = cursor.fetchone()
    if not index:
        return None
    config = get_config_by_id(int(index[0]))
    return config
```

main.py

```
import sys

from PyQt5.QtWidgets import QApplication

from UI import MainWindow
import db


def main():
    db.setup()
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())


if __name__ == '__main__':
    main()
```

ДОДАТОК Б Презентація





Постановка задачі

Задачею дипломної роботи є розробка програмного забезпечення, яке реалізуватиме генерування часових рядів вжитку серверних ресурсів програмними продуктами для їх оптимального розподілу між серверами.
Для досягнення мети потрібно виконати наступні задачі:

- проаналізувати методи генерування часових рядів;
- обрати модель для генерування;
- реалізувати алгоритм генерування часових рядів;
- відобразити результати у вигляді графіків.

Аналіз методів генерування часових рядів

ARIMA

ARIMA (AutoRegressive Integrated Moving Average) моделює часові ряди на основі попередніх значень. Цей метод підходить для генерування часових рядів із здебільшого стабільною та лінійною природою, проте не зовсім підходить для генерування навантаження, оскільки воно може змінюватися стрибкоподібно.

Аналіз методів генерування часових рядів

RNN

Рекурентні нейронні мережі (RNN) мають довгострокову пам’ять, що дозволяє виявляти та моделювати довгострокові залежності в часових рядах.
Проте, навантаження мікросервісами на серверні ресурси може змінюватися незалежно від попередніх значень.
Також RNN потребують великої кількості тренувальних даних та великих обчислювальних ресурсів.

Аналіз методів генерування часових рядів

TimeGAN

TimeGAN поєднують можливості генеративних моделей (GAN) та рекурентних нейронних мереж (RNN) для генерування часових рядів. Генерують реалістичні дані, враховуючи довгострокові залежності та сезонність даних.
Такий підхід вимагає доволі складної реалізації та великої кількості якісних вхідних даних.

Проєктування системи

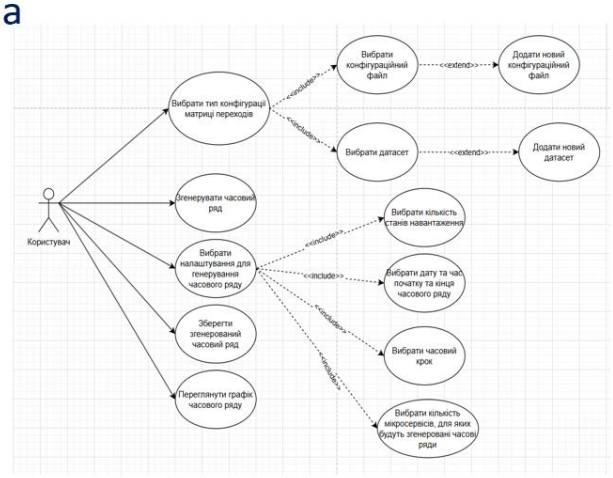
Ланцюги Маркова

Ланцюги Маркова реалізують математичну модель, яка описує систему, що переходить із одного стану в інший у межах певного набору станів. Ключова особливість ланцюгів Маркова – це властивість Маркова, що означає: ймовірність переходу до наступного стану залежить лише від поточного стану і не залежить від послідовності попередніх подій.

$$P = \begin{pmatrix} 0.4 & 0.3 & 0.2 & 0.1 \\ 0.2 & 0.5 & 0.2 & 0.1 \\ 0.1 & 0.3 & 0.4 & 0.2 \\ 0.1 & 0.1 & 0.4 & 0.4 \end{pmatrix}$$

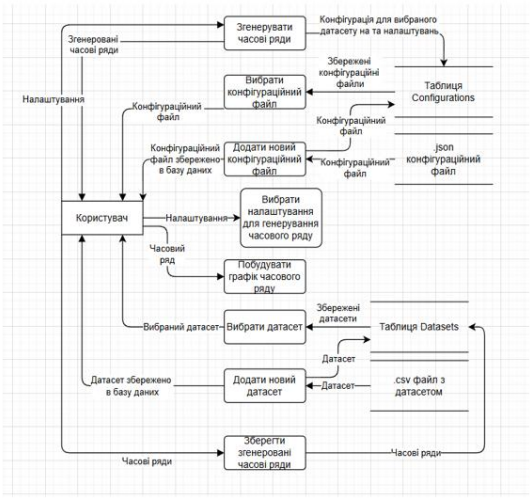
Проєктування системи

Use Case діаграма



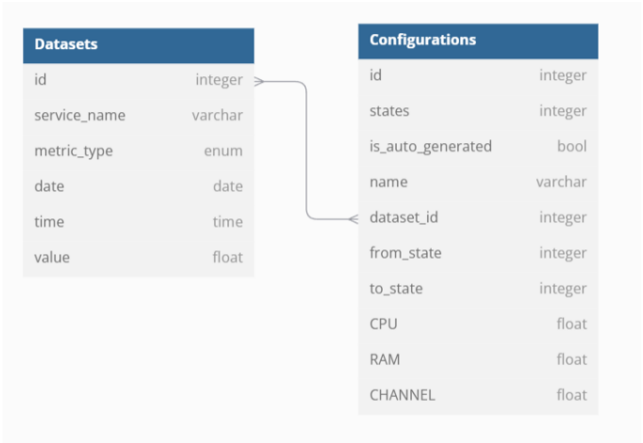
Проєктування системи

DFD діаграма



Проєктування системи

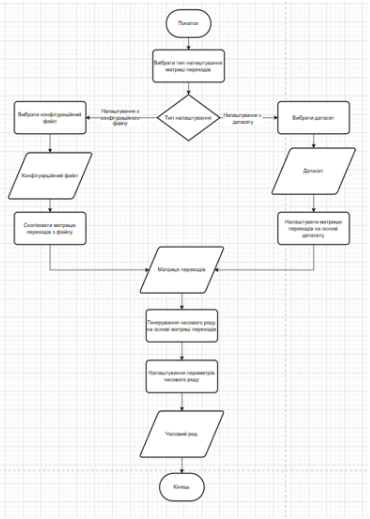
Структура бази даних



Реалізація системи

Алгоритм роботи системи:

- вибір типу налаштування матриці переходів (датасет або конфігураційний файл);
- вибір відповідного файлу для налаштування матриці переходів;
- налаштування матриці переходів;
- генерування часових рядів на основі ймовірностей матриці переходів.



Реалізація системи

У процесі розробки ПЗ було використано мову програмування Python, та такі бібліотеки:

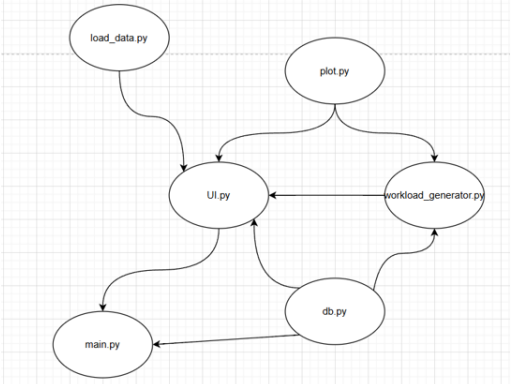
- csv – для роботи з .csv файлами;
- json – для роботи з .json файлами;
- matplotlib – для побудови графіків;
- numpy – для операцій над масивами;
- pandas – для обробки даних;
- sqlite3 – для роботи з базою даних Sqlite;
- PyQt – для графічного інтерфейсу;

Реалізація системи

Модулі системи

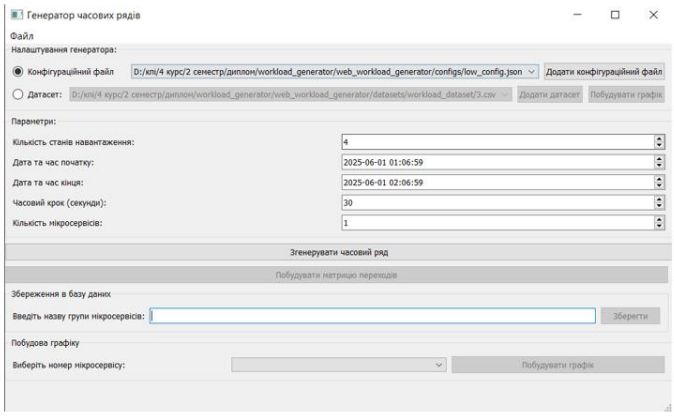
Система включає в себе шість модулів:

1. load_data.py – модуль завантаження файлів;
2. plot.py – модуль для побудови графіків;
3. db.py – модуль для роботи з БД;
4. workload_generator.py – модуль для генерування часових рядів;
5. UI.py – модуль реалізації інтерфейсу користувача;
6. main.py – модуль для запуску програми



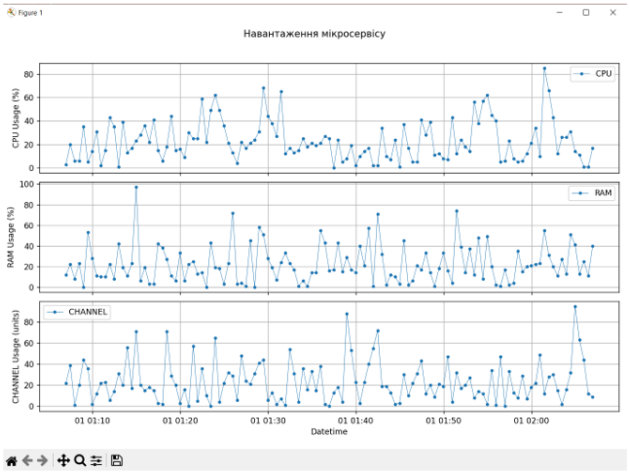
Реалізація системи

Головне вікно програми



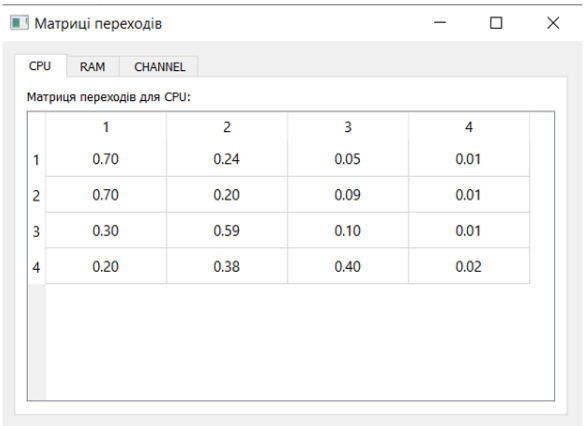
Реалізація системи

Графік згенерованого часового ряду



Реалізація системи

Матриця переходів



	1	2	3	4
1	0.70	0.24	0.05	0.01
2	0.70	0.20	0.09	0.01
3	0.30	0.59	0.10	0.01
4	0.20	0.38	0.40	0.02

Висновки

В результаті було реалізовано програмний продукт, який генерує часові ряди, що імітують навантаження мікросервісами на серверні ресурси, на основі ланцюгів Маркова.
Розроблене ПЗ дозволяє генерувати часові ряди на основі конфігураційних файлів з матрицями переходів, або ж на основі існуючих датасетів.
ПЗ передбачає налаштування параметрів для генерування часових рядів, а також їх збереження та відображення у вигляді графіків.

Національний технічний університет України
«Київський політехнічний інститутімені Ігоря Сікорського»

National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute"