НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

«На правах рукопису»
УДК 004.4`2

До захисту допущено:
В. о. завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
«___»_____2025 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних та інформаційних систем» зі спеціальності 121 “Інженерія програмного забезпечення” на тему: «Вебзастосунок для проєктування та аналізу систем масового обслуговування»

Виконав:
студент ІІ курсу, групи ІМ-41мп
Федяй Борислав Володимирович _____

Керівник:
Ас. каф. ОТ, PhD
Шульга Максим Володимирович _____

Консультант з нормоконтролю:
проф. каф. ОТ, д.т.н., професор
Жабін Валерій Іванович

Рецензент:
доц. каф. ІСТ, к.т.н.
Галушко Дмитро Олександрович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025

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

Факультет *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Рівень вищої освіти – другий (магістерський)

Спеціальність *121. Інженерія програмного забезпечення*
(код і назва)

Освітньо-професійна програма *Інженерія програмного забезпечення комп'ютерних та інформаційних систем*
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Михайло НОВОТАРСЬКИЙ
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Федяю Бориславу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема дисертації Вебзастосунок для проєктування та аналізу систем масового обслуговування

Науковий керівник дисертації Шульга Максим Володимирович, ас. каф. ОТ, PhD
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «6» листопада 2025 р. № 4841-с

2. Строк подання студентом дисертації 1 грудня 2025 р.

3. Об'єкт дослідження процес моделювання систем масового обслуговування у вебзастосунках

4. Предмет дослідження методи, алгоритми та архітектурні рішення для реалізації вебзастосунку моделювання СМО

5. Перелік завдань, які потрібно розробити: проведення аналізу теоретичних основ та існуючих програмних рішень для моделювання СМО, обґрунтування вибору технологій розробки, реалізування серверної частини застосунку на мові Rust із використанням фреймворку Rocket, проведення тестування ефективності та порівняльного аналізу продуктивності системи, оформлення результатів дослідження та формулювання висновків щодо доцільності використання запропонованого підходу.

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1.	Ас. каф. ОТ, PhD Шульга М. В.		
2.	Ас. каф. ОТ, PhD Шульга М. В.		
3.	Ас. каф. ОТ, PhD Шульга М. В.		
4.	Ас. каф. ОТ, PhD Шульга М. В.		
Нормоконтроль	проф., д.т.н., професор Жабін В. І.		

7. Дата видачі завдання 1 вересня 2025 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1.	Затвердження теми дослідження. Визначення предмету дослідження	1.09.25-7.09.25	
2.	Теоретичний аналіз предметної області. Формування науково-методичної основи майбутнього застосунку	8.09.25-21.09.25	
3.	Аналіз існуючого застосунку та обґрунтування переходу на Rust	22.09.25-28.09.25	
4.	Розробка та реструктуризація архітектури серверної частини	29.09.25-05.10.25	
5.	Реалізація серверної частини застосунку	06.10.25-19.10.25	
6.	Розробка та вдосконалення клієнтської частини	20.10.25-09.11.25	
7.	Тестування застосунку та порівняння оптимізації	10.11.25-13.11.25	
8.	Розробка стартап-проекту	14.11.25-18.11.25	
9.	Оформлення магістерської дисертації.	19.11.25-24.11.25	

Студент Борислав ФЕДЯЙ _____
(підпис)

Науковий керівник дисертації Максим ШУЛЬГА _____
(підпис)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Вебзастосунок для проєктування та аналізу систем масового обслуговування

студентом: Федяєм Бориславом Володимировичем

Робота складається зі вступу та чотирьох розділів. Загальний обсяг роботи: 79 аркушів основного тексту, 16 ілюстрацій, 26 таблиць. При підготовці використовувалася література з 17 різних джерел.

Актуальність. У сучасних умовах стрімкого розвитку інформаційних технологій та зростання обсягів даних питання ефективного моделювання процесів обслуговування запитів у складних системах набуває особливої важливості. Системи масового обслуговування (СМО) широко застосовуються в телекомунікаціях, транспорті, виробничих процесах та IT-сфері, де необхідно забезпечити оптимальний розподіл ресурсів і зменшення часу очікування.

Із розвитком високонавантажених вебсистем та розподілених обчислень зростає потреба у створенні сучасних інструментів моделювання, що поєднують високу точність, продуктивність і зручність використання. Саме тому розробка вебзастосунку для моделювання СМО із використанням сучасних технологій програмування, таких як Rust, є актуальним науково-прикладним завданням, що спрямоване на підвищення ефективності досліджень та оптимізації реальних систем обслуговування.

Мета і завдання дослідження. Метою магістерської роботи є розробка та дослідження вебзастосунку для моделювання систем масового обслуговування, який забезпечує високу продуктивність, надійність і гнучкість при побудові та аналізі моделей різної складності. Для досягнення поставленої мети вирішено такі завдання:

- провести аналіз теоретичних основ та існуючих програмних рішень для моделювання СМО;

- обґрунтувати вибір технологій розробки;
- реалізувати серверну частину застосунку на мові Rust із використанням фреймворку Rocket;
- провести тестування ефективності та порівняльний аналіз продуктивності системи;
- оформити результати дослідження та сформулювати висновки щодо доцільності використання запропонованого підходу.

Об’єкт дослідження – процес моделювання систем масового обслуговування у вебзастосунках.

Предмет дослідження – методи, алгоритми та архітектурні рішення для реалізації вебзастосунку моделювання СМО.

Методи досліджень. У роботі використано методи теорії масового обслуговування, теорії графів, імітаційного моделювання, а також методи аналізу складності алгоритмів і оцінювання ефективності програмних систем. Для перевірки працездатності системи застосовано експериментальні методи тестування та порівняння продуктивності.

Наукова новизна одержаних результатів роботи полягає у наступному:

- розроблено вебзастосунок для моделювання СМО, серверна частина якого реалізована на мові Rust, що забезпечує підвищену швидкодію та ефективне використання пам’яті;
- запропоновано архітектурне рішення, яке поєднує принципи композиції, сувору типізацію даних і асинхронну обробку запитів, що підвищує стабільність системи;
- виконано порівняльний аналіз продуктивності реалізації на Rust із попередньою версією на Python, що підтвердив триразове зростання швидкодії;
- удосконалено підхід до побудови моделювального середовища з можливістю інтерактивного формування та симуляції моделей СМО у вебінтерфейсі.

Особистий внесок здобувача. Усі етапи роботи – від аналізу предметної області до реалізації програмного продукту – виконані автором самостійно. Розроблено архітектуру вебзастосунку, реалізовано серверну частину системи на Rust, проведено тестування, аналіз продуктивності та оформлено результати дослідження. Формулювання мети, завдань і загальних напрямів дослідження здійснювалося спільно з науковим керівником.

Практична цінність. Отримані результати можуть бути використані:

- для створення та оптимізації систем аналізу та симуляції процесів масового обслуговування;
- у навчальних цілях для підготовки спеціалістів з моделювання, системного аналізу та програмної інженерії;
- як основа для подальшого розширення функціоналу, зокрема додавання розподілених обчислень чи візуалізації процесів у реальному часі.

Ключові слова. СИСТЕМА МАСОВОГО ОБСЛУГОВУВАННЯ, ВЕБЗАСТОСУНОК, ІМІТАЦІЙНЕ МОДЕЛЮВАННЯ, RUST, ROCKET, ПРОДУКТИВНІСТЬ, АРХІТЕКТУРА СИСТЕМИ.

ABSTRACT

The thesis consists of an introduction and four chapters. The total volume of the work includes 79 pages of main text, 16 figures, and 26 tables. The study is based on literature from 17 different sources.

Relevance. In the context of the rapid development of information technologies and the increasing volume of data, the problem of efficient modeling of request-servicing processes in complex systems becomes particularly important. Queueing systems (QS) are widely used in telecommunications, transportation, manufacturing processes, and the IT sector, where optimal resource allocation and minimization of waiting time are crucial.

With the growth of high-load web systems and distributed computing, there is a growing need for modern modeling tools that combine high accuracy, performance, and ease of use. Therefore, the development of a web application for QS modeling using modern programming technologies such as Rust is a relevant applied research task aimed at improving the efficiency of studies and optimizing real-world servicing systems.

Purpose and objectives of the study. The purpose of the master's thesis is to develop and investigate a web application for modeling queueing systems that ensures high performance, reliability, and flexibility in constructing and analyzing models of various complexities. To achieve this goal, the following tasks were accomplished:

- analysis of theoretical foundations and existing software solutions for QS modeling;
- justification of the choice of development technologies;
- implementation of the server side of the application in Rust using the Rocket framework;
- performance testing and comparative analysis of the system’s efficiency;

– preparation of research findings and formulation of conclusions regarding the applicability of the proposed approach.

Object of the study – the process of modeling queueing systems in web applications.

Subject of the study – methods, algorithms, and architectural solutions used to implement a QS modeling web application.

Research methods. The study employs methods of queueing theory, graph theory, simulation modeling, algorithmic complexity analysis, and performance evaluation of software systems. Experimental testing and comparative performance measurements were conducted to validate the system's operability.

Scientific novelty. The scientific novelty of the obtained results lies in the following:

– a web application for QS modeling was developed, with its server side implemented in Rust, ensuring increased performance and efficient memory usage;
– an architectural solution was proposed that combines composition principles, strict data typing, and asynchronous request processing, improving the stability of the system;
– a comparative performance analysis of the Rust implementation versus the previous Python version was conducted, demonstrating a threefold increase in processing speed;
– an improved approach to designing a modeling environment with interactive construction and simulation of QS models in a web interface was introduced.

Author's contribution. All stages of the work – from domain analysis to the implementation of the software product – were completed by the author. The architecture of the web application was designed, the server side was implemented in Rust, system testing and performance evaluation were carried out, and the research results were prepared. The formulation of the goals, objectives, and general research direction was conducted in cooperation with the scientific supervisor.

Practical value. The obtained results can be used:

- for developing and optimizing systems for analyzing and simulating queueing processes;
- in educational activities for training specialists in modeling, systems analysis, and software engineering;
- as a basis for further functional expansion, including the integration of distributed computing or real-time process visualization.

Keywords. QUEUEING SYSTEM, WEB APPLICATION, SIMULATION MODELING, RUST, ROCKET, PERFORMANCE, SYSTEM ARCHITECTURE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API	–	Application programming interface, прикладний програмний Інтерфейс.
СМО	–	Система масового обслуговування.
IT	–	Інформаційні технології.
JSON	–	JavaScript Object Notation.

ВСТУП

Стрімкий розвиток вебтехнологій та зростання обсягів інформації зумовлюють підвищення вимог до продуктивності, масштабованості та стабільності сучасних вебзастосунків. У більшості сфер, пов’язаних з обробкою великих потоків даних – телекомунікаціях, хмарних обчисленнях, транспортних та виробничих системах – важливим є точне моделювання процесів обслуговування запитів. Для таких задач широко застосовуються системи масового обслуговування (СМО), які дозволяють аналізувати динаміку потоку заявок, оцінювати навантаження на ресурси та оптимізувати їх використання.

Зростання складності моделей та необхідність обробки значних обсягів подій у симуляціях формують підвищені вимоги до швидкодії програмних засобів моделювання. Існуючі інструменти часто демонструють обмеження у продуктивності, зокрема при виконанні інтенсивних обчислень у реальному або наближеному до реального часі. Попередні програмні реалізації вебзастосунків для моделювання СМО, створені мовою Python, забезпечували функціональність, проте їх продуктивність у складних симуляціях виявлялася недостатньою для масштабних сценаріїв.

Однією з актуальних тенденцій розвитку високопродуктивних системних рішень є застосування мов програмування, що поєднують низькорівневу ефективність і безпечну роботу з пам’яттю. Мова програмування Rust є сучасним інструментом системного програмування, орієнтованим на створення надійних, безпечних і високопродуктивних програмних систем. Її використання дає змогу усунути низку обмежень, притаманних інтерпретованим мовам, зокрема витрати на динамічну типізацію та розподіл пам’яті під час виконання.

З огляду на зазначене, підвищення ефективності вебзастосунку для моделювання систем масового обслуговування шляхом переходу на Rust є обґрунтованим і актуальним завданням. Такий підхід дозволяє підвищити

швидкодію симуляцій, стабільність системи та ефективність використання апаратних ресурсів.

Метою магістерської роботи є підвищення продуктивності вебзастосунку для проєктування та аналізу СМО шляхом реалізації його серверної частини мовою Rust та дослідження впливу цього підходу на якість і швидкість моделювання.

Результати роботи можуть бути використані під час створення веборієнтованих інструментів симуляції, у навчальному процесі, а також у подальших дослідженнях, пов’язаних із моделюванням систем обслуговування, оптимізацією складних потокових систем та розробленням високопродуктивних вебсервісів.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Класифікація систем масового обслуговування

Система масового обслуговування (СМО) – це математична модель, що описує процес надходження, очікування та обслуговування великої кількості заявок у певній системі. У найзагальнішому вигляді СМО складається з джерела заявок, пристроїв або каналів обслуговування, а також черги, де заявки можуть очікувати на обслуговування у разі зайнятості всіх каналів. Схематичний вигляд структури СМО наведено на рис. 1.1.

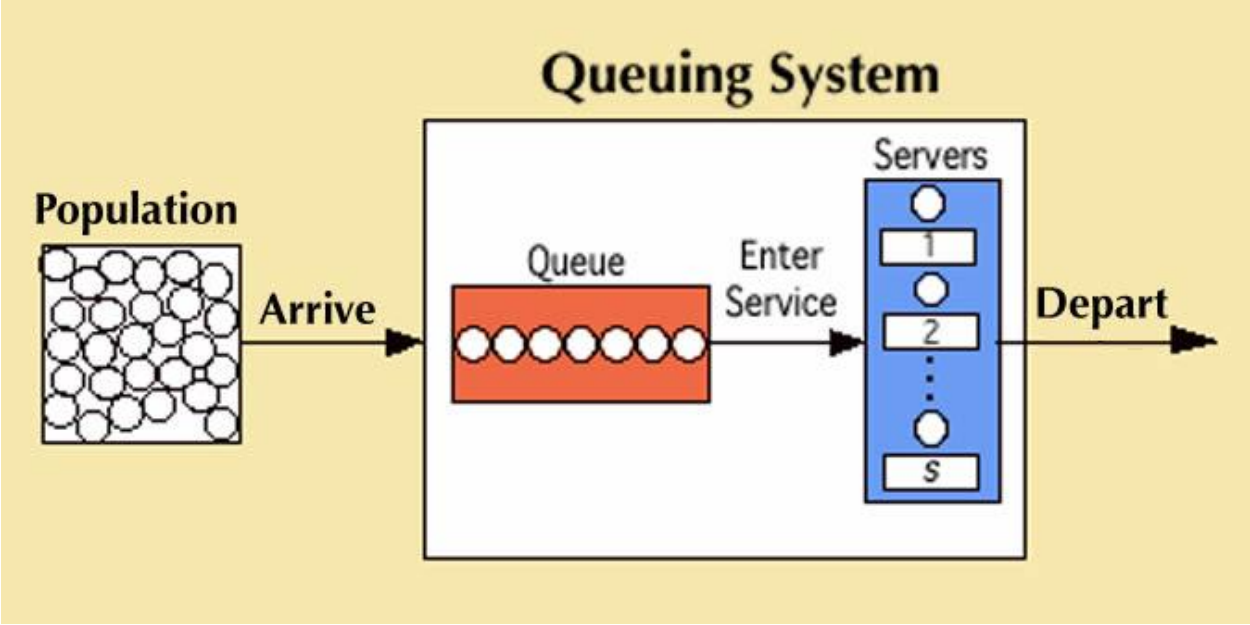


Рис. 1.1. Схематичний приклад процесу роботи СМО

Основною метою аналізу СМО є оптимізація процесів обслуговування, що дає змогу мінімізувати час очікування, підвищити пропускну здатність системи та, найголовніше, раціонально використовувати ресурси. Теорія масового обслуговування надає аналітичні та імітаційні методи для дослідження й прогнозування поведінки таких систем залежно від параметрів потоку заявок та характеристик каналів обслуговування.

СМО застосовуються у різних галузях [1]: у телекомунікаціях вони моделюють роботу серверів, маршрутизаторів і мережевих вузлів, де заявки – це пакети даних; у транспортних системах – це потоки автомобілів чи пасажирів, які обслуговуються на перехрестях чи вокзалах; у сфері IT – це запити користувачів до вебсервісів або баз даних. У кожному з цих випадків мета моделювання полягає в визначенні показників ефективності (пропускної здатності, часу очікування, ймовірності відмови тощо) та їх оптимізації.

З математичного погляду СМО часто описується за допомогою потоків подій і марковських процесів. Найпоширенішою є пуассонівська модель надходження заявок, у якій інтервали між подіями мають експоненційний розподіл. Для позначення типу СМО використовується нотація Кендалла (A/B/C/K/N/D) [2], де символи визначають типи розподілів, кількість каналів, розмір черги, кількість джерел та дисципліну обслуговування. Така формалізація дозволяє класифікувати й аналізувати широкий спектр систем – від простих одноканальних до багаторівневих мереж обслуговування. Крім того, СМО також класифікують за їх ознаками [3]:

1) за кількістю каналів обслуговування;
    a. одноканальні – обслуговування здійснює один канал (наприклад, одна каса);
    b. багатоканальні – обслуговування здійснює кілька каналів одночасно;
2) за наявністю та типом черги;
    a. з відмовами – заявка втрачається, якщо не отримала обслуговування;
    b. з очікуваннями – заявки можуть очікувати своєї черги обслуговування. У низці моделей черга може мати обмеження;
3) за дисципліною обслуговування;
    a. First In, First Out (FIFO);
    b. Last In, First Out (LIFO);
    c. Service In Random Order (SIRO);

d. за пріоритетом;

4) за кількістю фаз;
    a. однофазний – етап обробки заяви один;
    b. багатофазний – кілька послідовних етапів;
5) за структурою системи;
    a. відкриті – отримують заявки із зовнішнього середовища:
    b. закриті – кількість заявок у системі обмежена;
6) за режимом роботи;
    a. стаціонарні – параметри з часом не змінюються;
    b. нестаціонарні – параметри з часом, навпаки, змінюються.

У межах цього проєкту СМО характеризується за наступними параметрами:

1) багатоканальність – користувач може вказувати кількість обслуговуючих каналів;
2) наявність обмежених черг (хоча їх можна також зробити й ефективно нескінченними);
3) обслуговування в порядку SIRO – оскільки система не «бачить» самих заявок (вони представлені лише у вигляді загального числа заявок) і має інформацію лише про їх кількість у черзі, порядок обслуговування не має значення;
4) однофазність – один канал виконує лише один етап обробки заявки;
5) відкрита структура – робота моделі обмежується не кількістю заявок, а віртуальним часом роботи;
6) стаціонарність;

Відповідно, саме цей тип системи буде використано як базову модель для аналізу ефективності роботи вебзастосунку та подальшої оптимізації.

1.2 Основні характеристики та параметри систем масового обслуговування

Основна мета аналізу СМО полягає в визначенні ефективності функціонування системи, її пропускної здатності та якості обслуговування користувачів. Знання характеристик СМО дозволяє кількісно оцінювати її роботу, порівнювати різні конфігурації систем, а також здійснювати оптимізацію параметрів для підвищення ефективності обслуговування. Залежно від задачі, параметри СМО поділяють на вхідні, структурні та вихідні (результативні) характеристики [4].

1.2.1 Вхідні параметри

До вхідних параметрів належать характеристики потоку заявок, які надходять до системи:

- інтенсивність надходження заявок (λ) – середня кількість заявок за одиницю часу;
- закон розподілу інтервалів між надходженнями заявок – хоча найчастіше використовується експоненційний розподіл, що відповідає пуассонівському потоку заявок, можуть застосовуватись і інші розподіли: нормальній, рівномірний, за Ерлангом тощо;
- тип потоку заявок;
  - простий (пуассонівський);
  - регулярний (рівномірний);
  - груповий;
  - із пріорітетами.

1.2.2 Структурні параметри

Структурні характеристики визначають організацію процесу обслуговування:

- кількість каналів обслуговування (n) – кількість сутностей, які одночасно можуть обслуговувати заявки;
- інтенсивність обслуговування (μ) – середня кількість заявок, що можуть бути оброблені одним каналом за одиницю часу;
- розподіл часу обслуговування;
- наявність черги;
- дисципліна обслуговування.

1.2.3 Вихідні (результативні) параметри

Результативні показники відображають ефективність роботи системи та якість обслуговування заявок:

- коефіцієнт завантаження системи (ρ) – визначає частку часу, протягом якої система зайнята обслуговуванням заявок;

  Коефіцієнт завантаження системи обчислюється за формулою (1.1):

$$\rho = \frac{\lambda}{n\mu} \tag{1.1}$$

- імовірність простою системи (P_0) – імовірність того, що всі канали обслуговування вільні;
- імовірність відмови (P_o) – імовірність, що заявка не отримає обслуговування через зайнятість усіх каналів і переповнення черги;
- середня довжина черги (L_q) – очікувана кількість заявок, що очікують обслуговування;
- середній час очікування в черзі (W_q) (1.2);

$$W_q = \frac{L_q}{\lambda} \tag{1.2}$$

- середній час перебування заявки в системі (W) (1.3);

$$W = W_q + \frac{1}{\mu} \tag{1.3}$$

- пропускна здатність системи (A) – кількість заявок, які фактично обслуговуються за одиницю часу ($A \leq \lambda$);
- імовірність очікування обслуговування (P_w) – частка заявок, які потрапляють до черги;
- коефіцієнт ефективності використання ресурсів (η) (1.4).

$$\eta = \frac{A}{n\mu} \tag{1.4}$$

1.3 Методи аналізу та моделювання систем масового обслуговування

Ефективність роботи будь-якої системи, у якій відбувається обслуговування великої кількості запитів, безпосередньо залежить від здатності цієї системи швидко реагувати на зміну навантаження. Аналіз СМО ж, в свою чергу, дозволяє кількісно оцінити поведінку таких систем, виявити вузькі місця, визначити оптимальну кількість ресурсів і визначити компроміс між якістю обслуговування та витратами.

Практичний інтерес до СМО пояснюється тим, що в більшості сучасних технічних і інформаційних систем процеси обслуговування є стохастичними, а саме випадковими. Потоки запитів користувачів, пакети даних у мережі, дзвінки до контакт-центру або клієнти в банківському відділенні надходять у непередбачуваний момент часу, а тривалість їх обробки також випадково варіюється.

Розглянемо типову телекомунікаційну мережу, у якій користувачі надсилають запити до сервера. Кожен запит потрапляє у чергу, очікуючи на вільний канал обслуговування. Якщо потік збільшується, зростає черга, час відповіді системи погіршується, а при перевантаженні деякі запити можуть втрачатися.

У цьому випадку аналіз СМО дає змогу визначити відповіді на ключові питання:

- скільки каналів обслуговування потрібно для забезпечення заданого рівня якості сервісу;
- яка ймовірність відмови при певному навантаженні;
- як зміна інтенсивності потоку або швидкості обслуговування впливає на час;
- як розподілити ресурси між кількома вузлами мережі для мінімізації затримок.

На основі таких розрахунків з’являється можливість оптимізувати роботу, наприклад, серверів, балансувальників навантаження та мережевих каналів, не вдаючись до надмірного збільшення апаратних ресурсів.

Безпосередньо дослідження СМО здійснюється за допомогою методів, які можна умовно поділити на аналітичні та імітаційні (моделювальні). Вибір методу залежить від складності системи, наявності аналітичних залежностей та вимог до точності отриманих результатів.

1.3.1 Аналітичні методи

Аналітичні методи базуються на строгих математичних моделях, що описують поведінку СМО за допомогою апарату теорії ймовірностей, марковських процесів і диференціальних рівнянь. Ці методи дозволяють отримати точні формули для основних характеристик системи – середньої довжини черги, часу очікування, ймовірності відмови та коефіцієнта завантаження.

- марковські моделі та процеси народження-смерті – одні із базових інструментів, які описують зміну кількості заявок у системі з часом [2];

Наприклад, для системи типу M/M/1 диференціальні рівняння Колмогорова мають вигляд (1.5):

$$\frac{dP_n(t)}{dt} = \lambda P_{n-1}(t) + \mu P_{n+1}(t) - (\lambda + \mu)P_n(t) \qquad (1.5)$$

де $P_n(t)$ – ймовірність того, що в системі перебуває n заявок у момент часу t,

λ – інтенсивність потоку надходжень,

μ – інтенсивність обслуговування.

Рівноважні (стаціонарні) ймовірності станів визначаються як (1.6):

$$P_n = (1-\rho)\rho^n \tag{1.6}$$

$$\rho = \frac{\lambda}{\mu} < 1 \tag{1.7}$$

І на основі цих залежностей вже виводяться показники продуктивності системи:

- середня кількість заявок у черзі (1.8);

$$L_q = \frac{\rho^2}{1-\rho'} \tag{1.8}$$

- середній час очікування (1.9) тощо.

$$W_q = \frac{\rho}{\mu(1-\rho)} \tag{1.9}$$

- матрично-аналітичні методи – застосовуються для більш складних моделей, де потоки не мають експоненційного розподілу [5]. Вони дозволяють розв’язувати моделі типу M/PH/1, PH/M/1, BMAP/PH/1 тощо, де часи надходження або обслуговування описуються фазовими розподілами. Основна ідея полягає у використання матричних рівнянь, які описують динаміку ймовірностей станів системи;
- наближені аналітичні моделі – використовуються, коли точне рішення неможливе. Найвідомішими з них є:
  - формула Кінґмена – для оцінки середнього часу у системах типу G/G/1;
  - наближення до інтенсивного руху – застосовується для аналізу перевантажених систем, де коефіцієнт завантаження $\rho \to 1$;
  - метод еквівалентного потоку – використовується для спрощення аналізу складних мереж черг.

Аналітичні методи є ефективними для теоретичного аналізу й побудови еталонних моделей, проте вони мають обмеження при моделюванні систем зі складною структурою або нетиповими розподілами.

1.3.2 Імітаційне моделювання

Імітаційне (або дискретно-подійне) моделювання вважається універсальним підходом дослідження СМО, який дозволяє вивчати динаміку роботи системи шляхом просування подій у часі [6]. У моделі описується послідовність подій – надходження заявки, початок і завершення обслуговування, вихід із системи. Для кожної події визначаються її час, параметри та вплив на стан системи.

Імітаційні моделі забезпечують можливість врахувати:

- довільні закони розподілу для надходжень і обслуговування;
- пріорітети заявок;
- обмежену кількість ресурсів;
- складні топології систем.

І саме тому цей підхід широко застосовується у комп’ютерному моделюванні, транспортних системах тощо, коли аналітичний опис є надмірно складним або неможливим.

1.4 Висновки до розділу 1

У першому розділі магістерської дисертації було проведено теоретичний аналіз предметної області, пов’язаної з СМО, що є основою для опису й оптимізації процесів обробки запитів у різних сферах – від телекомунікацій і вебсерверів до транспортних та виробничих систем.

Було визначено, що СМО є стохастичною моделлю, яка описує процеси надходження, очікування та обслуговування заявок у багатоканальному середовищі. Розглянуто класифікацію СМО та їх характеристики.

Окрему увагу приділено методам аналізу та моделювання СМО. Показано, що аналітичні підходи дозволяють отримувати точні теоретичні результати для простих моделей. Водночас імітаційне моделювання дозволяє досліджувати складні системи з довільними законами розподілу та взаємодіями між елементами.

Таким чином, у результаті аналізу предметної області сформовано науково-методичне підґрунтя для подальшої розробки та оптимізації вебзастосунку, який реалізує інструменти проєктування та аналізу СМО.

Додатково, проведене дослідження дозволило виділити ключові фактори ефективності систем масового обслуговування та визначити обмеження, пов’язані з класичними підходами до їх моделювання. Це створює основу для обґрунтованого вибору технологій та алгоритмів, які забезпечують баланс між точністю симуляції, швидкодією та гнучкістю системи, що є критично важливим для практичної реалізації вебзастосунку. Такий підхід гарантує, що розроблений інструмент не лише відображає теоретичні аспекти СМО, а й ефективно використовується для аналізу реальних процесів у різних галузях.

РОЗДІЛ 2
СПОСОБИ ПІДНЯТТЯ ЕФЕКТИВНОСТІ ЗАСТОСУНКУ

2.1 Огляд поточних проблем у застосунку

Поточна версія застосунку функціонує коректно й стабільно виконує всі поставлені завдання. Однак, у процесі первинної розробки було прийнято низку архітектурних рішень, які, хоча й забезпечили швидку реалізацію проєкту, частково обмежили його подальший потенціал. Інакше кажучи, застосунок працює добре, але має значний резерв для оптимізації.

Основним напрямом удосконалення є оптимізація алгоритму симуляції моделі СМО. Цей алгоритм побудовано на основі одного головного циклу, який виконується доти, доки віртуальний годинник не досягне встановленої часової межі. Кожна ітерація циклу містить послідовність таких кроків:

1) визначення найближчої події за часом;
2) оновлення поточного часу для всіх елементів системи та корекція їх статистики;
3) передача заявок від каналів, у яких щойно відбулася подія;
4) логування події;
5) перевірка перетину часової межі та, за потреби, повторення циклу.

З погляду логічної структури алгоритм уже наближений до мінімально можливої абстракції. Значна його частина – це обробка відносно коротких масивів, виконання базових арифметичних операцій та запис подій у лог. Тому можливості для розпаралелювання процесів практично відсутні. Подальше покращення асимптотичної складності вимагало б глибокої реконструкції серверної частини застосунку, що суттєво ускладнило б подальшу підтримку системи.

Водночас є більш перспективний шлях – оптимізація не структури алгоритму, а середовища його виконання. Поточна реалізація побудована

мовою програмування Python, яка забезпечує високу швидкість розробки та гнучкість завдяки своїй абстракції [7]. Однак ця ж абстракція має зворотний бік: вона створює додаткові накладні витрати на рівні інтерпретації, управління пам’яттю та типізації, що суттєво обмежує ефективність при обробці великої кількості подій. Python чудово підходить для побудови прототипів і реалізації бізнес-логіки, проте для обчислювально-інтенсивних задач, таких як симуляція складних моделей СМО, він не завжди здатен забезпечити необхідну продуктивність.

2.2 Вибір технологій для оптимізації

Під час пошуку способів підвищення ефективності системи доцільно розглянути можливість реалізації серверної частини на інших мовах програмування. Якщо спускатися на нижчий рівень абстракції, природними кандидатами стають Java та C#. Ці мови давно зарекомендували себе як надійні інструменти для розробки складних корпоративних систем і мають розвинені екосистеми.

Однак, попри свої переваги, Java та C# залишаються високорівневими мовами, які мають певні накладні витрати через віртуальні машини (JVM та .NET CLR відповідно) і складні системи типізації [8]. У задачах, де ключовими є швидкодія і мінімальні затримки, таки рішення не завжди можуть забезпечити оптимальний результат.

Наступним кроком є розгляд мов нижчого рівня – C та C++, які традиційно вважаються «золотим стандартом» ефективності. Вони дозволяють безпосередньо працювати з пам’яттю, управляти потоками виконання і створювати максимально оптимізований код. Саме тому C/C++ десятиліттями були основними мовами для реалізації високопродуктивних обчислювальних систем і симуляційних моделей.

Проте, попри свою швидкодію C та C++ мають суттєві недоліки у контексті сучасних веборієнтованих систем. Їх складність у керуванні пам’яттю та громіздкі засоби компіляції роблять процес розробки тривалим і потенційно ризикованим. Крім того, створення на них повноцінних вебзастосунків вимагає значних додаткових зусиль, адже більшість сучасних фреймворків і бібліотек орієнтовані на мови вищого рівня.

До того ж упродовж останнього десятиліття на сцені з’явилися нові системні мови, які прагнуть поєднати найкраще з обох світів – ефективність C/C++ і зручність високорівневих технологій. Найяскравішим представником цього нового покоління стала мова програмування Rust, яка пропонує сучасний, безпечний і високопродуктивний підхід до системного та веброзроблення.

2.2.1 Мова програмування Rust

Серед сучасних системних мов програмування особливе місце посідає Rust – мова, яка поєднує продуктивність і контроль C/C++ [9] із безпечністю та виразністю високорівневих мов. Її поява стала відповіддю на багаторічну проблему розробки ефективного, але водночас надійного програмного забезпечення без класичних помилок керування пам’яттю, що характерні C і C++.

Rust з самого початку був спроєктований як мова для реального промислового використання, а не як академічний експеримент. Його створення підтримала компанія Mozilla, а нині він активно застосовується Microsoft, Amazon, Google, Dropbox, Cloudflare, Discord, Facebook (Meta) та іншими технологічними гігантами [10]. Наприклад:

- Amazon Web Services (AWS) використовує Rust у критичних компонентах своєї хмарної інфраструктури для підвищення безпеки й швидкодії;

- Microsoft інтегрує Rust у ядро Windows для усунення вразливостей, пов’язаних із небезпечним доступом до пам’яті;
- Cloudlfare та Discord частково переписали свої високонавантажені сервіси на Rust, отримавши суттєве зниження затримок і споживання ресурсів.

Такий рівень довіри з боку промисловості свідчить, що Rust уже давно вийшов за межі експериментів і став практичним інструментом для продакшену.

З технічного погляду, Rust пропонує сучасну систему керування пам’яттю без Garbage Collector. Завдяки механізмам ownership, borrowing та lifetimes компілятор гарантує відсутність витоків пам’яті, подвійного звільнення чи висячих вказівників – і робить це на етапі компіляції, без додаткових витрат під час виконання [11]. Це дозволяє отримати ефективність на рівні C/C++, але без типових помилок цих мов.

Компілятор Rust – один із найсуворіших серед сучасних мов: він змушує розробника писати код, який не може призвести до помилок керування пам’яттю або race conditions у багатопотоковому середовищі. Попри те, що поріг входу для початківців може бути вищим, досвідчені розробники швидко відзначають, що Rust зменшує кількість дрібних помилок і спрощує підтримку коду на великих проєктах, що підтверджує стрімкий ріст його популярності (рис 2.1).

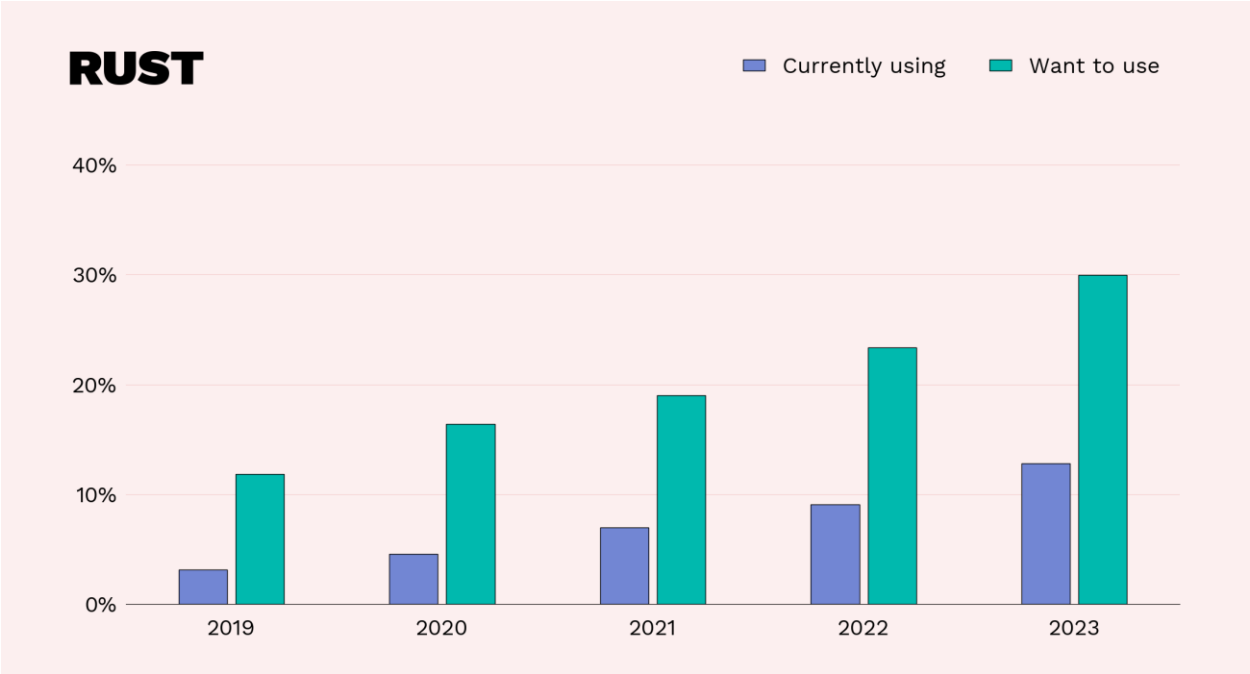


Рис. 2.1. Ріст популярності Rust за даними опитувань Stack Overflow ([17])

Крім безпеки, Rust вирізняється швидкістю компіляції та високою продуктивністю виконання. Він компілюється у машинний код через LLVM (той самий бекенд, що використовує Clang/LLVM для C++), що забезпечує мінімальний runtime і продуктивність, близьку до «чистого» C [12]. Завдяки цьому Rust став популярним у сфері системного програмування, розробки вбудованих систем, блокчейн-платформ [13], а також високопродуктивних вебсерверів.

2.2.2 Вебфреймворк Rocket

Ще однією причиною популярності Rust є розвинена екосистема та зручні інструменти розробки. В цьому сенсі він дуже схожий на Python.

Одним із найпопулярніших фреймворків для веброзробки на Rust є Rocket – сучасний, безпечний і високопродуктивний інструмент, створений спеціально для побудови серверних застосунків і REST API. Rocket вирізняється строгою типобезпекою та мінімалістичним синтаксисом, який дозволяє описувати маршрути, запити й відповіді у максимально зрозумілій формі, зберігаючи при цьому всю потужність компілятора.

Однією з ключових особливостей Rocket є використання статичної перевірки типів на рівні маршрутизації. Наприклад, якщо маршрут очікує числовий параметр, компілятор не дозволить запустити сервер із невідповідним типом запиту. Це забезпечує надзвичайно високий рівень надійності, який практично неможливо досягти у фреймворках динамічних мов (привіт, FastAPI). І як приємний бонус, Rocket також підтримує асинхронну обробку запитів, що робить його придатним для побудови високонавантажених систем реального часу.

Окрім Rocket, у спільноті Rust активно розвиваються й інші вебфреймворки, зокрема:

- Actix Web – один із найшвидших HTTP-фреймворків у світі [14];
- Axum – сучасний фреймворк, орієнтований на простоту, модульність і сумісність з екосистемою Tokio;
- Warp – легковаговий фреймворк із фокусом на функціональному стилі.

Щоб наочно порівняти популярність цих фреймворків разом із Rocket, у таблиці 2.1 наведено основні метрики: кількість завантажень із crates.io, популярність на GitHub та активність розробників.

Таблиця 2.1

Популярність веб-фреймворків для Rust

Метрика	Rocket	Actix Web	Axum	Warp
Кількість завантажень на crates.io за весь час	9,179,994	48,679,939	180,547,646	31,900,951
Кількість завантажень на crates.io за останні 90 днів	2,134,694	7,988,464	37,362,096	4,352,933
Кількість зірок на GitHub (тис.)	25.4	23.8	23.4	10.1
Кількість користувачів фреймворку на GitHub (тис.)	33.9	71.6	73	30

Таблиця 2.1 (продовження)

Популярність веб-фреймворків для Rust

Кількість учасників розробки	308	386	397	173

Попри наявність альтернатив, саме Rocket було обрано для реалізації серверного компонента цього проєкту, оскільки він:

- забезпечує найкраще співвідношення простоти синтаксису та безпеки коду;
- має чітку структуру додатків і зручну інтеграцію з системами шаблонізації;
- надає продуману документацію і стабільну API.

У контексті даної роботи Rocket виступає як оптимальний компроміс між швидкодією (котра на рівні запитів великої ролі взагалі не грає), надійністю та зручністю розробки, дозволяючи ефективно реалізувати серверну частину вебзастосунку без втрати продуктивності чи безпеки.

2.3 Висновки до розділу 2

Проведений аналіз показав, що попри стабільну роботу поточного застосунку, його архітектура та вибір технологій обмежують потенціал подальшої оптимізації. Основна обчислювальна складність зосереджена у симуляційному алгоритмі, який, бувши логічно спрощенім до мінімуму, не має значного простору для структурного вдосконалення чи розпаралелювання. Це зумовлює необхідність зміщення фокуса оптимізації від самого алгоритму до середовища його виконання.

Використання Python забезпечило швидку розробку та гнучкість у створенні прототипу, однак інтерпретована природа цієї мови створює суттєві обмеження під час обробки великої кількості подій. Для задач, пов’язаних із

моделюванням імітаційних процесів та СМО, де критичними є час відгуку й обчислювальна ефективність, така технологічна база виявляється недостатньою.

Порівняльний аналіз альтернативних рішень продемонстрував, що хоча Java та C# мають розвинені екосистеми й високий рівень абстракції, їх віртуальні середовища (JVM та .NET CLR) створюють додаткові накладні витрати. C та C++ залишаються еталонами продуктивності, проте складність керування пам’яттю ускладнюють розробку веборієнтованих систем.

На цьому тлі мова програмування Rust постає як сучасна альтернатива, що поєднує продуктивність рівня C/C++ із надійністю та безпечністю високорівневих мов. Її механізм управління пам’яттю без Garbage Collector, система ownership/borrowing та строгий компілятор гарантують відсутність типових помилок керування пам’яттю без втрати швидкодії. Активне використання Rust у промислових продуктах Microsoft, Amazon, Cloudflare та інших компаній підтверджує його зрілість для продакшену.

У межах веброзробки фреймворк Rocket забезпечує ефективний, безпечний і зручний спосіб реалізації серверної частини застосунку. Серед альтернативних фреймворків (Actix Web, Axum, Warp) саме Rocket було обрано як оптимальний компроміс між продуктивністю, зручністю розробки та стабільністю API.

Таким чином, у межах даного дослідження вибір мови Rust і фреймворку Rocket є обґрунтованим кроком для підвищення ефективності системи. Це рішення дозволяє зберегти архітектурну логіку застосунку, суттєво знизити накладні витрати виконання й забезпечити вищий рівень надійності при розробці серверної частини вебзастосунку для моделювання СМО.

РОЗДІЛ 3
ПРАКТИЧНА РОЗРОБКА ТА ТЕСТУВАННЯ ОНОВЛЕННОГО ЗАСТОСУНКУ

3.1 Архітектура оновленого застосунку

Архітектура оновленого вебзастосунку зберігає логічну структуру початкової системи, однак також вона зазнала низки адаптацій, пов’язаних із переходом на нове технологічне середовище. Окрім очевидних зміни мови програмування та відповідних бібліотек, ключовими стали дві структурні модифікації:

1) заміна наслідування композицією – оскільки Rust не підтримує класичне наслідування, характерне C-подібним мовам, основна парадигма об’єктного розширення була реалізована через композицію. Тобто тепер не дочірній клас є розширенням батьківського, а батьківська структура містить у собі вкладені типи, що визначають поведінку конкретного компонента;
2) зміна шаблонізатора HTML із Jinja2 на Tera – через перехід із FastAPI (Python) на Rocket (Rust) довелося змінити шаблонізатор. Обидві технології мають схожий синтаксис, тому перехід був відносно безболісним.

3.2 Огляд розробленого програмного забезпечення

Оновлений вебзастосунок для проєктування та аналізу систем масового обслуговування реалізує повний цикл роботи з моделлю – від створення структури СМО до проведення симуляції та візуалізації результатів. Система

має модульну архітектуру, що дає змогу ізолювати окремі елементи функціоналу та забезпечує гнучкість у подальшому розвитку проєкту.

Користувацька частина представлена вебінтерфейсом, через який користувач може створювати, редагувати та аналізувати моделі СМО. Інтерфейс побудований за принципом інтерактивного конструктора, що дає змогу формувати структуру СМО у вигляді графа, визначати параметри елементів (інтенсивність потоків, кількість каналів, черги тощо), визначати потенціальні вузькі місця і запускати симуляцію безпосередньо в браузері.

Серверна частина відповідає за обробку запитів та логіку симуляції. Вона реалізована на мові Rust та фреймворку Rocket. Сервер приймає запити від клієнта, виконує симуляцію процесів у СМО, формує звіти про статистичні показники та повертає результати у форматі JSON.

Основні функціональні можливості розробленого застосунку:

- створення моделей СМО довільної складності;
- локальне збереження моделей для майбутнього їх завантаження до застосунку;
- симуляція роботи з урахуванням параметрів кожного з елементів;
- графічне відображення результатів у вигляді графіків;
- логування перших та останніх N подій симуляції для подальшого аналізу;
- вивід витраченого часу та максимального використання пам’яті застосунком під час симуляції;
- завчасне прогнозування можливих вузьких місць у моделі ще до початку симуляції.

У наступних розділах буде більш конкретно розглянуто усі поточні зміни у проєкті.

3.2.1 Компонент графічного конструктора

Безпосередньо сама сторінка фактично не змінилась. З візуальних змін є лише спрощення списку елементів, що доступні на панелі зліва (рис 3.1-3.2). Це дозволить відійти від обмеженого, з погляду семантики, функціоналу застосунку до сфери загального вжитку, а не тільки IT-сфери.

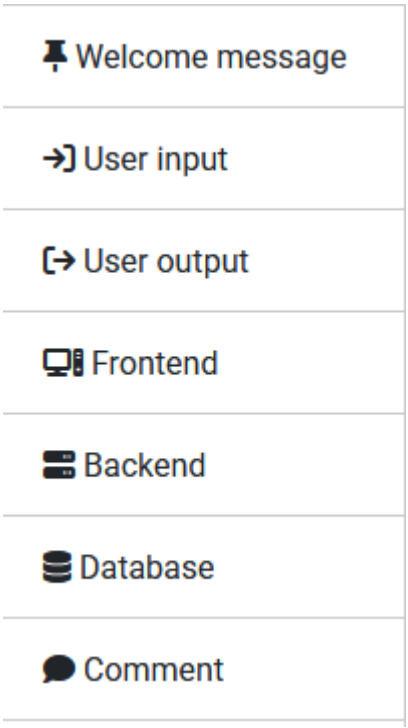


Рис. 3.1. Список елементів у попередній версії застосунку

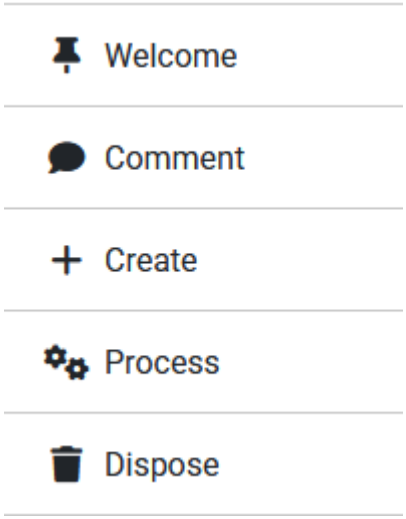


Рис. 3.2. Список елементів у поточній версії застосунку

Усі інші зміни стосуються або розбиття файлів зі стилями, або незначними змінами логіки для відповідання новому фреймворку.

Лише можна виділити одну серйозну зміну, пов’язану з оптимізацією даних в запиті на початок симуляції моделі. Попередньо, через відсутність компоненту підготовки запиту (рис 3.3), на сервер відправлялось багато даних не актуальних для побудови моделі та її симуляції. В результаті це зменшує кількість пам’яті, потрібної для роботи застосунку, а також і так мізерне навантаження на мережу.

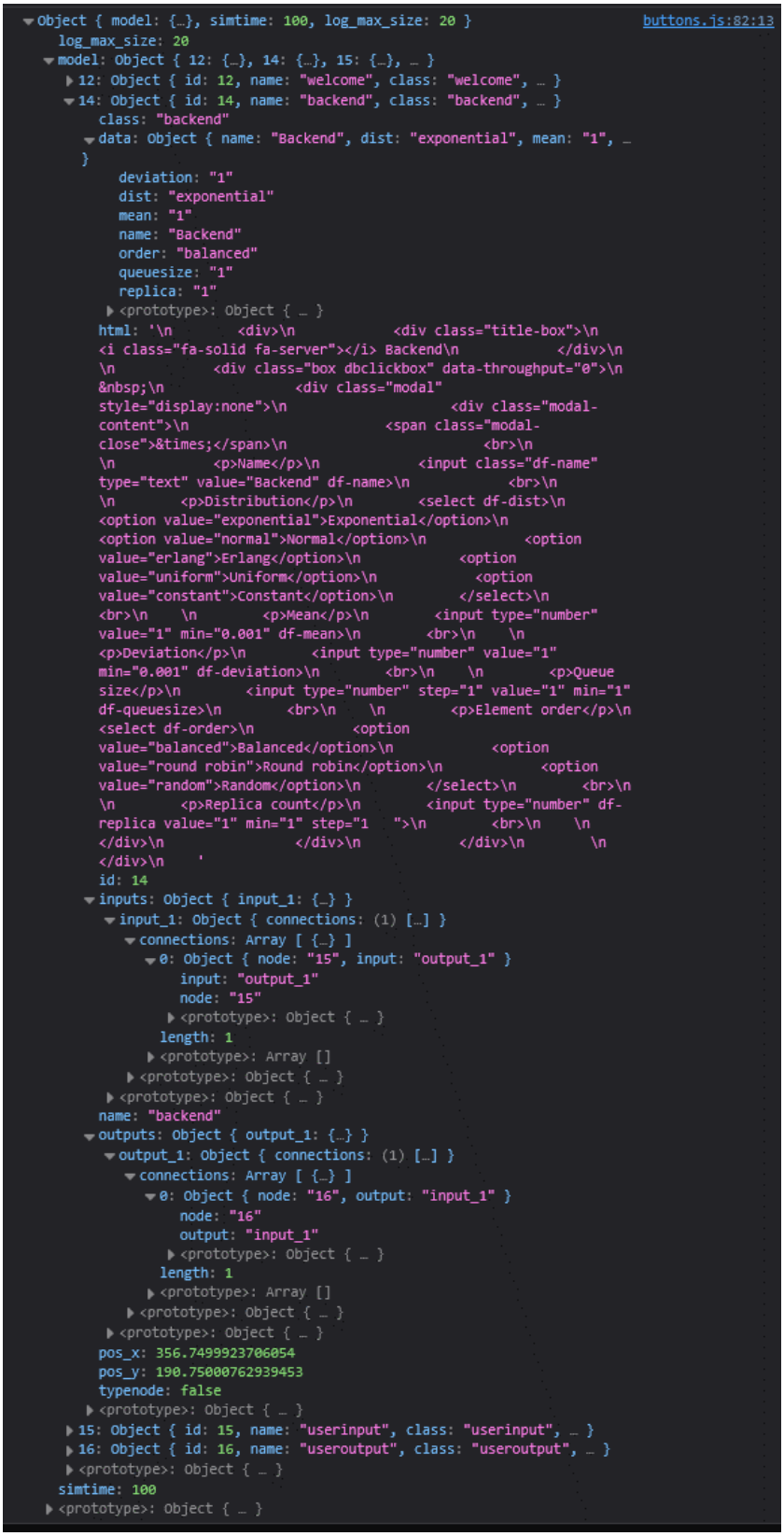


Рис. 3.3. Структура одного елементу, що відправляється на сервер у попередній версії застосунку

Звернути увагу треба на величезне поле «html», дуже глибоку вкладеність даних про входи та виходи елементів та наявність елементу «welcome», що не бере участі в симуляції взагалі.

```
function prepareModel() {
    let model = editor.export()["drawflow"]["Home"]["data"]
    for (const [id, e] of Object.entries(model)) {
        // clear redundant fields
        delete e.html
        delete e.id
        delete e.name
        delete e.pos_x
        delete e.pos_y
        delete e.typenode

        // edit element data
        switch (e.class) {
            case "create":
            case "process": …
            case "dispose": …
            default:
                // delete everything else
                delete model[id]
                break
        }
        // resturcture IO
        e.inputs = Object.values(e.inputs)
            .flatMap(input => input.connections.map(c => Number(c.node)));
        e.outputs = Object.values(e.outputs)
            .flatMap(output => output.connections.map(c => Number(c.node)));
    }

    return model
}
```

Рис. 3.4. Функція для спрощення та фільтрації даних, що відправляються на сервер у поточній версії застосунку

```
▼Object { model: {…}, simtime: 100, log_max_size: 20 }                buttons.js:81:13
   log_max_size: 20
  ▼model: Object { 14: {…}, 15: {…}, 16: {…} }
    ▼14: Object { data: {…}, class: "create", inputs: [], … }
       class: "create"
      ▼data: Object { name: "Create", dist: "exponential", mean: 1, … }
          deviation: 1
          dist: "exponential"
          mean: 1
          name: "Create"
          order: "balanced"
          queuesize: 1
          replica: 1
        ▶<prototype>: Object { … }
      ▼inputs: Array []
          length: 0
        ▶<prototype>: Array []
      ▼outputs: Array [ 15 ]
          0: 15
          length: 1
        ▶<prototype>: Array []
      ▶<prototype>: Object { … }
    ▶15: Object { data: {…}, class: "process", inputs: (1) […], … }
    ▶16: Object { data: {…}, class: "dispose", inputs: (1) […], … }
    ▶<prototype>: Object { … }
     simtime: 100
   ▶<prototype>: Object { … }
```

Рис. 3.5. Структура одного елементу, що відправляється на сервер у поточній версії застосунку

3.2.2 Компонент формування графу систем масового обслуговування

З погляду логіки, компонент працює так само як і раніше, але реалізація в нього відрізняється кардинально. Зміни пов’язані з рядом наступних причин:

1) спрощення тіла запиту (рис 3.3-3.5);
2) структура тіла запиту тепер не аморфна, а має чітку та однакову структуру для усіх елементів моделі – робота тепер йде не з асоціативним масивом на кшталт JSON, а чіткою структурою (рис 3.6);

```rust
#[derive(Deserialize)]
#[serde(crate = "rocket::serde")]
pub struct SimRequest {
    model: HashMap<String, ElementInfo>,
    simtime: f64,
    log_max_size: u64,
}

#[derive(Deserialize)]
pub struct ElementInfo {
    pub class: String,
    pub data: ElementData,
    pub inputs: Vec<usize>,
    pub outputs: Vec<usize>,
}

#[derive(Deserialize)]
pub struct ElementData {
    pub deviation: f64,
    pub dist: String,
    pub mean: f64,
    pub name: String,
    pub order: String,
    pub queuesize: u32,
    pub replica: u32,
}
```

Рис. 3.6. Структури, що описують тіло запиту на ендпоінті «/simulate»

3) інша структура класів на стороні сервера – оскільки наслідування було замінено на композицію, це потягнуло за собою і зміни в структурі проєкту;
4) елементи тепер «пов’язуються» між собою не вказівниками, а ідентифікаторами – технічно, це продовження попереднього пункту, але з погляду архітектури застосунку це дуже суттєво.

Вдаватись у деталі реалізації безпосередньо елементів не має сенсу, оскільки змін дуже багато, але розглядати їх окремо від логіки компонента симуляції СМО сенсу мало. Але якщо підбити підсумки по цьому компоненту, то його поведінка стала простішою, більш зрозумілою та передбачливою, бо в ній тепер менше перевірок та циклів. Також, завдяки тому, що робота йшла з чітко описаною та, знову ж таки, спрощеною структурою даних, безпосередньо розробка йшла значно краще.

3.2.3 Компонент симуляції систем масового обслуговування

У компоненті симуляції СМО змін дуже багато і фактично усі вони викликані відмовою від наслідування та зміною системи пов’язування елементів між собою. Окрім неуникних змін також багато зусиль було вкладено в оптимізацію окремих процесів симуляції. Зручніше усього буде покроково розглянути зміни через головну функцію цього компоненту – simulate():

1) найпершою оптимізацією, ще навіть до виклику simulate(), є ініціалізація масивів для логів зі заздалегідь визначеною довжиною. Оскільки логи оновлюються на кожній ітерації симуляції, це зекономить на постійному перерозподілу пам'яті;
2) першим кроком симуляції є ініціалізація двох масивів заданої довжини для усіх ідентифікаторів елементів, котрі в цей момент передають заявки між собою. Раніше в цьому не було потреби, оскільки за передачу заявок відповідали самі елементи, а не симулятор;
3) далі визначається час найближчої події та оновлюється годинник у симулятора. Жодних суттєвих змін;
4) далі відбувається передача заявок між елементами. Як вже було зазначено, раніше це відбувалось на рівні елементів, а тепер – на рівні симулятору. Також через це, вибір отримувача заявки також тепер відбувається у симуляторі, а не в окремому елементі. Семантично, цей

крок навіть було спрощено завдяки тому, що це відбувається в один цикл ітерування повного списку елементів замість двох;

5) і наостанок, логування події. З погляду логіки, все залишилось на своїх місцях, але замість роботи з обгорткою String натомість тут ведеться робота з базовими типами даних, що також покращує оптимізацію як пам’яті, так і швидкості.

Підбиваючи підсумки, логічно алгоритм змінено не було, однак він був покращений та спрощений як семантично, так і на рівні управління пам'яттю.

3.2.4 Компонент виводу результатів роботи системи масового обслуговування

Компонент виводу результатів змінився як функціонально, так і візуально. По-перше, тепер графіків усього п’ять (рис 3.7):

1) ймовірність втрат;
2) загальна кількість втрат;
3) середня довжина черги;
4) час простою;
5) кількість оброблених заявок;

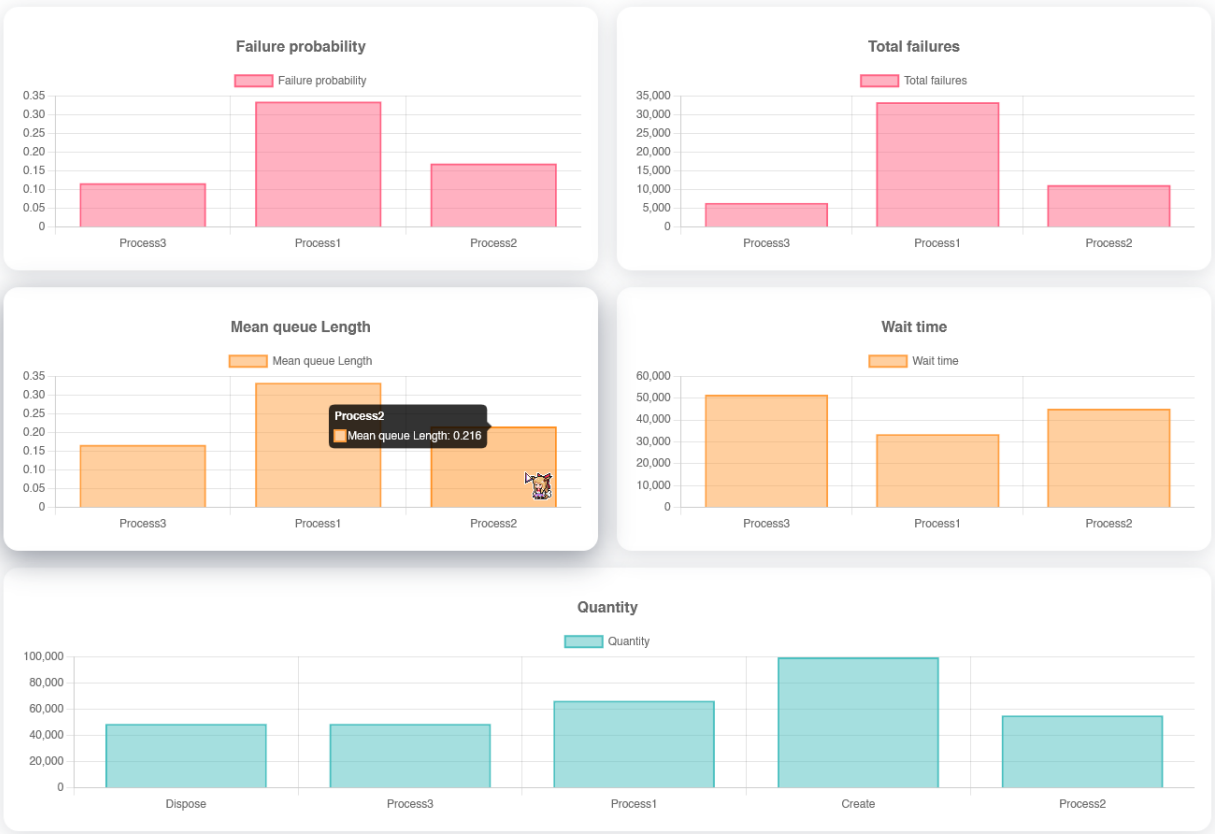


Рис. 3.7. Графіки із результатами

Раніше графіків було лише 4, і на місці часу простою стояла кількість оброблених заявок. Зміну зумовила фільтрація усіх постачальників та споживачів заявок від відображення верхніх чотирьох графіків, оскільки вони завжди мали б нульові або максимальні значення. До того ж завдяки цьому навігація по довшому графіку кількості оброблених заявок також стала зручнішою.

Наступною зміною стало створення вікна зі статистикою обчислення симуляції (рис. 3.8). В першу чергу воно було додано для зручності тестування, але користувачам ця інформація також може стати у нагоді.

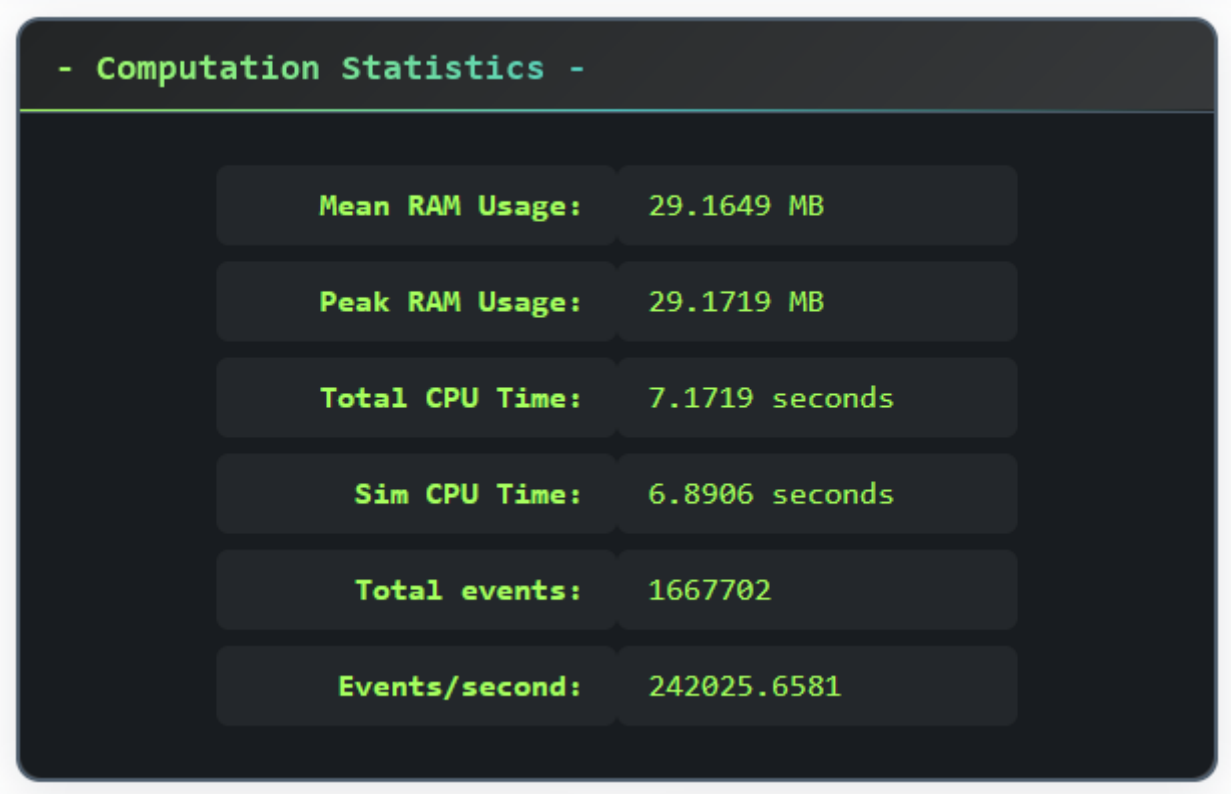


Рис. 3.8. Вікно зі статистикою роботи застосунку по обчисленню симуляції

І, наостанок, абсолютно косметично було покращено вікна з логами – тепер вони загалом більші та візуально кращі (рис 3.9).

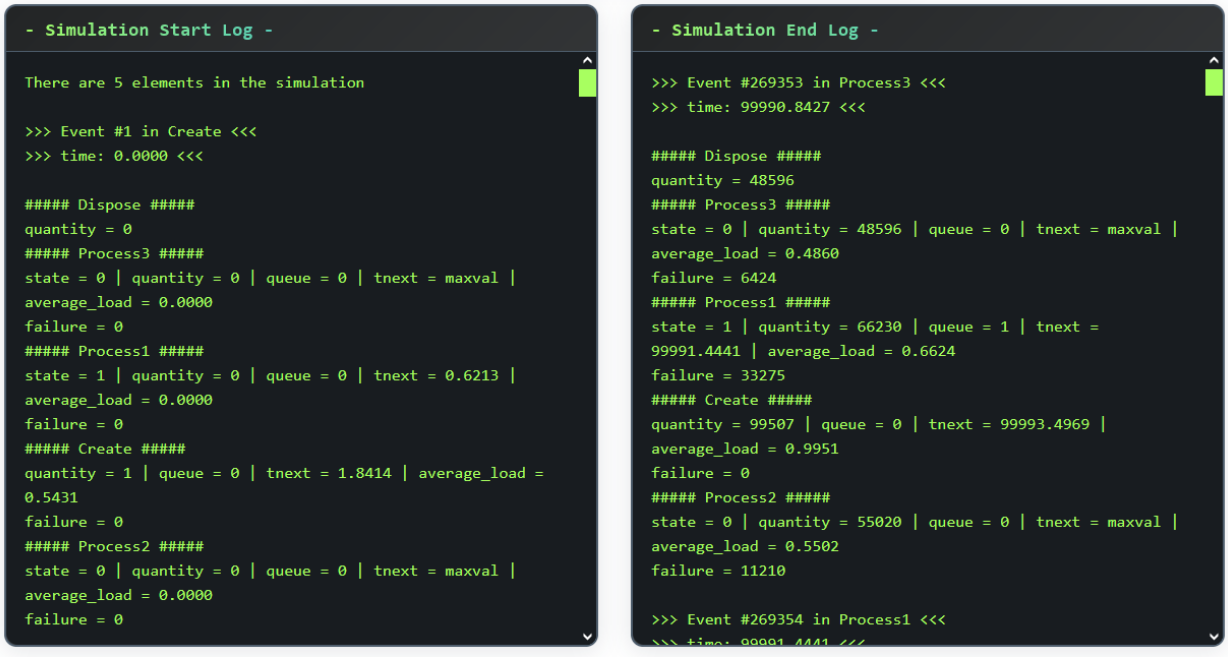


Рис. 3.9. Вікна з логами

3.3 Тестування ефективності оновленого застосунку

Метою тестування є визначення фактичного покращення швидкодії та оцінка впливу переходу на мову Rust на загальну ефективність системи. Окрім продуктивності, також буде проведено порівняння якості коду між попередньою та новою реалізаціями, щоб підтвердити, що перепис не призвів до деградації структури та підтримуваності проєкту.

3.3.1 Порівняння оптимізації

Усі експерименти проводилися на одному й тому самому пристрої, в однакових умовах та на ідентичних моделях. Оптимізація оцінювалася за статистичними показниками, які програма повертала після завершення симуляції (рис 3.7), а саме:

- середня зайнятість оперативної пам’яті;
- час обробки усього запиту на симуляцію (CPU time [15]);
- час обчислення симуляції (CPU time);
- кількість оброблених подій;
- швидкість обчислення подій (за секунду).

У даній реалізації модель СМО представляється у вигляді орієнтованого графа, що складається з вузлів трьох типів: постачальників, обробників та знищувачів. Алгоритм симуляції є повністю лінійним: кожна ітерація циклу виконує фіксований набір операцій незалежно від розміру графа. Відповідно, зі збільшенням кількості вузлів змінюється лише загальна кількість ітерацій, але не обчислювальна складність кожної окремої ітерації.

Модель СМО у цьому проєкті представлена у вигляді орієнтованого графа із трьох типів вузлів – постачальників, обробників та знищувачів. Основний цикл симуляції має лінійну часову складність O(N), де N – кількість елементів моделі. Це зумовлено необхідністю на кожній ітерації знаходити найближчу подію серед усіх елементів та оновлювати їх статистику. Решта

операцій виконуються над обмеженим числом вузлів і не впливають на загальну складність.

Таким чином, збільшення розміру графа впливає на тривалість симуляції лінійно і не призводить до появи нових типів навантаження або змін у структурі обчислень. Тому для порівняння ефективності реалізацій на різних мовах достатньо обмежитися трьома моделями різної розмірності (рис. 3.9-3.11), що репрезентують малі, середні та великі системи. Це дозволить зберегти експеримент контрольованим, а результати – статистично стабільними.

Перша модель (рис. 3.10) являє собою просту модель з розгалуженням, де постачальник за збалансованою логікою розподіляє заявки – отримувач визначається за найбільш вільною чергою. Під «збалансованою логікою» мається на увазі, що на етапі вибору наступного елементу при передачі заявки, система намагатиметься вибрати елемент з найбільш вільною чергою.

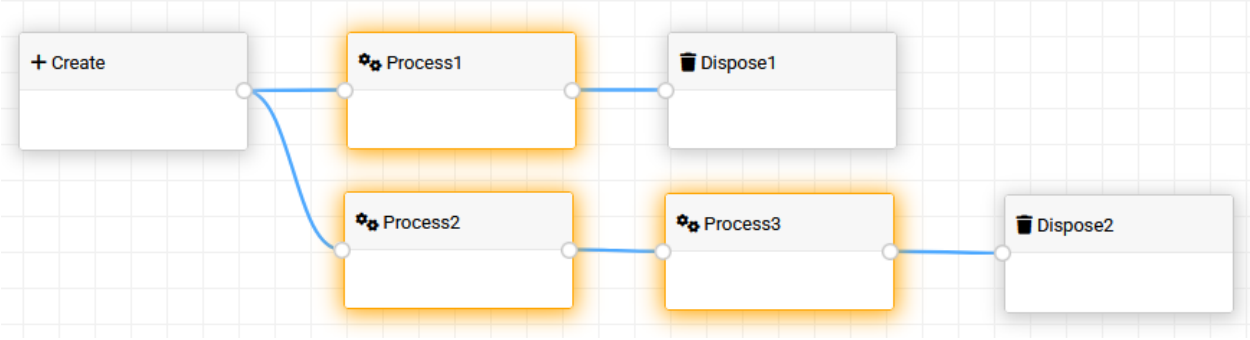


Рис. 3.10. Мала модель СМО

Моделювання проводилось упродовж 500,000 одиниць часу із 20 записами у логах подій. Результати наведено в таблиці 3.1.

Таблиця 3.1

Результати моделювання малої моделі СМО

Статистика	Python	Rust	Покращення (%)
Середнє завантаження пам’яті (MB)	62.1367	24.2266	256.48
Загальний час обробки запиту (с)	21.2656	6.8125	312.16

Таблиця 3.1 (продовження)

Результати моделювання малої моделі СМО

Час симуляції (с)	21.2651	6.7344	315.77
Кількість ітерацій	1149728	1142384	-
Ітерації / с	54066.4291	169634.1174	313.75

Друга модель - це модель середнього розміру (рис 3.11). В ній, окрім виходів, також є циклічні зв'язки, що повертають заявки у початок моделі.

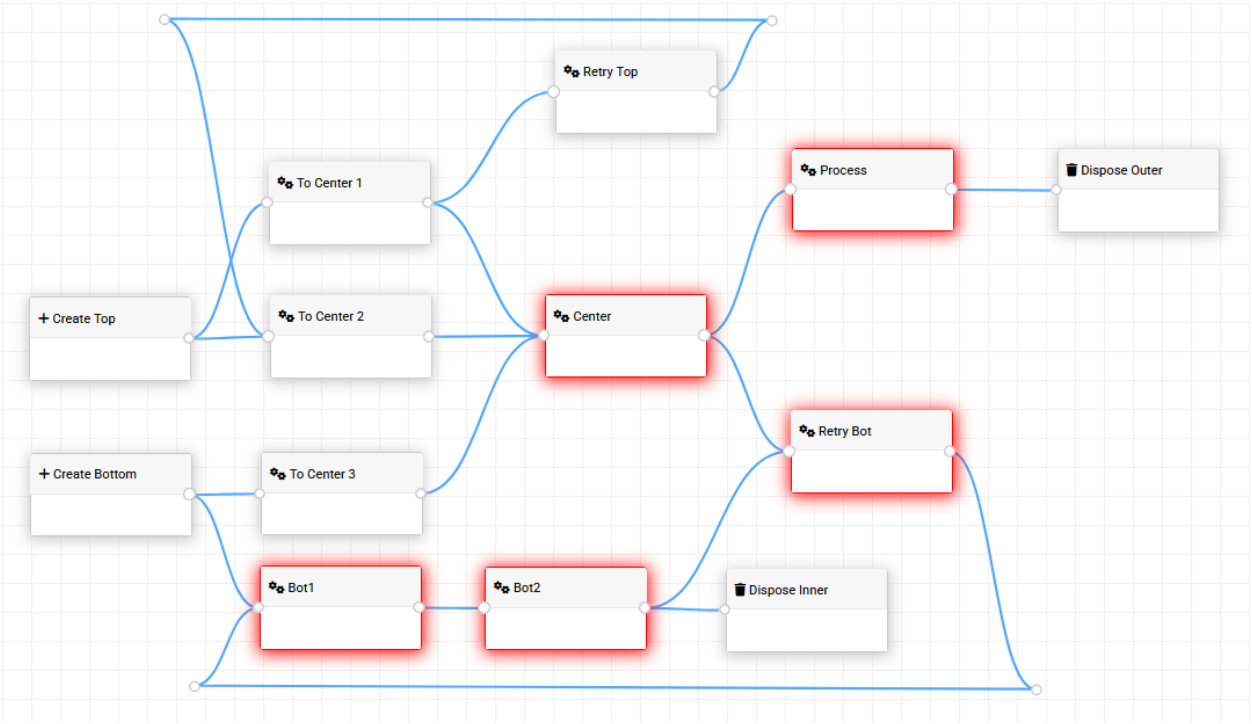


Рис. 3.11. Середня модель СМО

Моделювання проводилось упродовж 500,000 одиниць часу із 20 записами у логах подій. Результати наведено в таблиці 3.2.

Таблиця 3.2

Результати моделювання середньої моделі СМО

Статистика	Python	Rust	Покращення (%)
Середнє завантаження пам'яті (MB)	59.9847	21.0098	285.51
Загальний час обробки запиту (с)	134.0502	38.3594	349.46

Таблиця 3.2 (продовження)

Результати моделювання середньої моделі СМО

Час симуляції (с)	134.0494	38.1094	351.75
Кількість ітерацій	3581103	3139872	-
Ітерації / с	26714.8038	82391.0111	308.41

Третя і найбільша модель (рис. 3.12) характеризується великою кількістю випадкових зв'язків та елементів загалом. Кожна заявка успішно виходить з системи лише в 1/8 випадків.

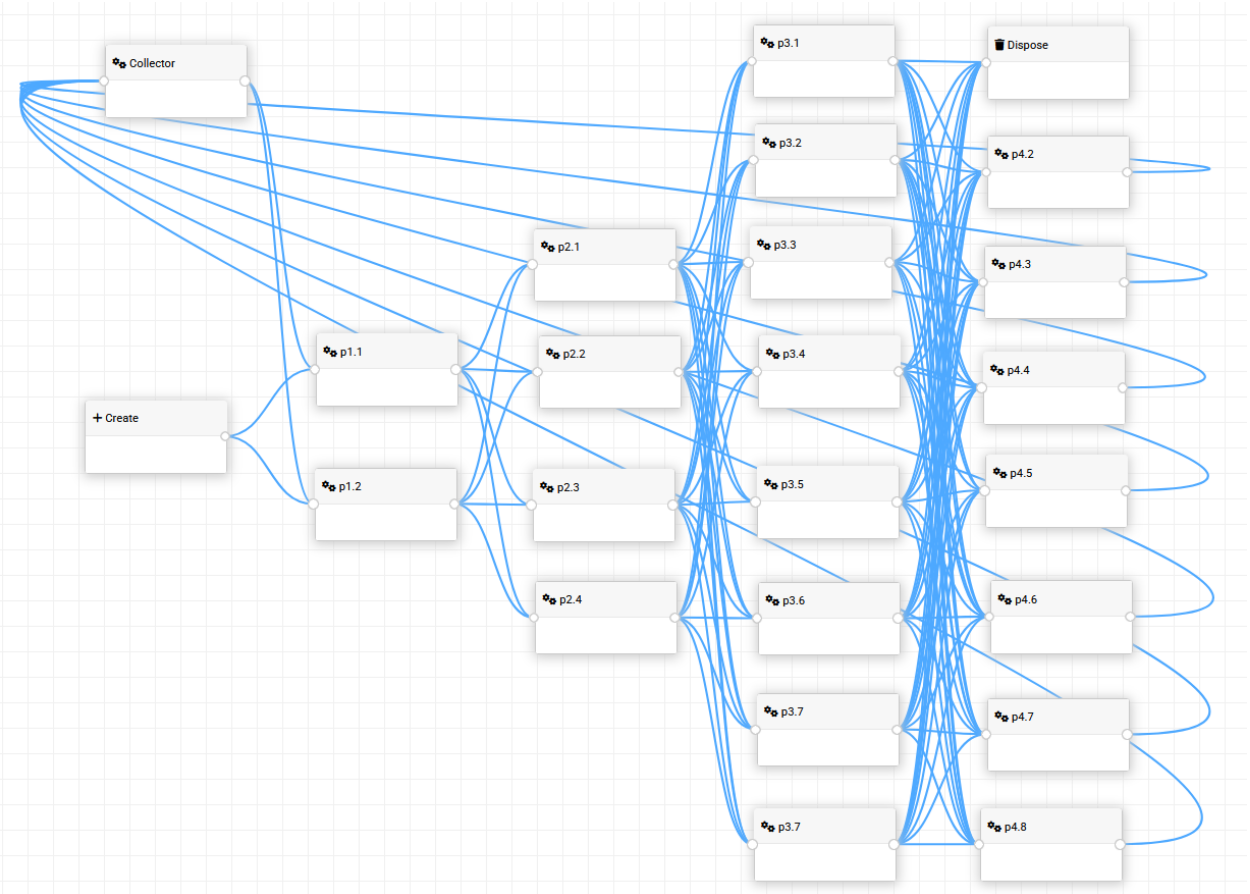


Рис. 3.12. Велика модель СМО

Моделювання проводилось упродовж 100,000 одиниць часу із 20 записами у логах подій. Результати наведено в таблиці 3.3.

Таблиця 3.3

Результати моделювання великої моделі СМО

Статистика	Python	Rust	Покращення (%)
Середнє завантаження пам’яті (MB)	63.0391	24.4102	258.25
Загальний час обробки запиту (с)	62.7225	24.1875	259.32
Час симуляції (с)	62.7212	24.125	259.98
Кількість ітерацій	1114607	1118759	-
Ітерації / с	17770.8157	46373.4301	260.95

Проведене тестування показує, що переписування серверної частини застосунку з Python на Rust дало відчутне підвищення ефективності за всіма вимірюваними параметрами. У середньому, продуктивність зросла більш ніж утричі, що підтверджується зменшенням часу виконання моделі.

В усіх трьох експериментальних моделях Rust стабільно демонструє скорочення часу симуляції у 2.6-3.5 рази та зниження використання оперативної пам'яті приблизно у 2.5-3 рази порівняно із Python-реалізацією. Отримані результати узгоджуються з очікуваннями щодо продуктивності:

- відсутність інтерпретатора та динамічної типізації у Rust мінімізує накладні витрати на виконання коду;
- компіляція у машинний код через LLVM забезпечує ефективне використання процесорних ресурсів;
- жорстка система керування пам’яттю знімає необхідність у Garbage Collector, що знижує пікові навантаження.

Таким чином, реалізація серверної частини симулятора на Rust довела свою доцільність для задач, де критичними є швидкодія, стабільність часу відгуку та ефективність використання ресурсів. Це підтверджує гіпотезу про те, що перехід до нижчого рівня абстракції може дати суттєве покращення без

погіршення якості коду чи втрати гнучкості в подальшому розширенні системи.

3.3.2 Порівняння якості коду

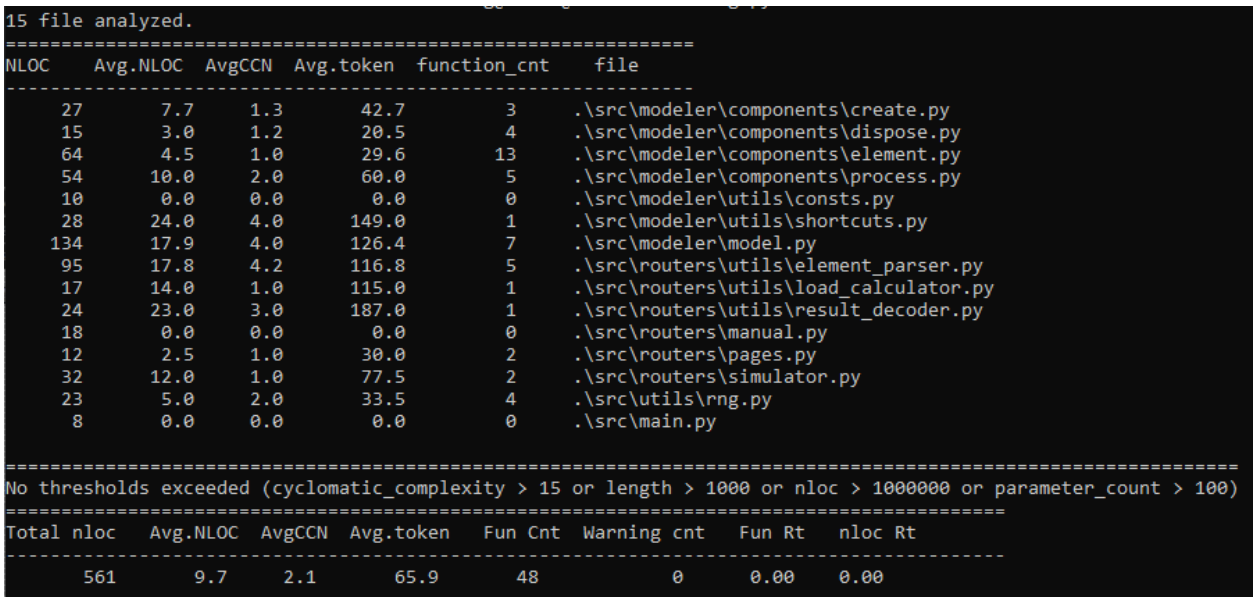
Окрім вимірювання продуктивності симуляцій, було проведено порівняльний аналіз якості реалізації серверної частини обох версій застосунку. Метою цього етапу є визначення, чи вплинув перехід на інше середовище виконання на структурну складність програмного коду, а також чи зберігся баланс між продуктивністю та зрозумілістю реалізації.

Для збору було обрано мово-агностичний застосунок Lizard, який аналізує вихідний код та надає кількісні показники складності. Оцінювання виконувалося лише для серверної частини застосунку, оскільки саме цей компонент зазнав основних змін у процесі оптимізації.

Lizard формує такі ключові характеристики коду:

- nloc – кількість рядків коду без коментарів, що відображає загальний обсяг логіки;
- CCN (Cyclomatic Complexity Number або цикломатична складність) – характеризує кількість незалежних шляхів виконання в програмі [16];
- token count – кількість токенів у функціях, що слугує індикатором їхньої деталізації;
- parameter count – кількість параметрів у функціях, що визначає ступінь зв’язності модулів.

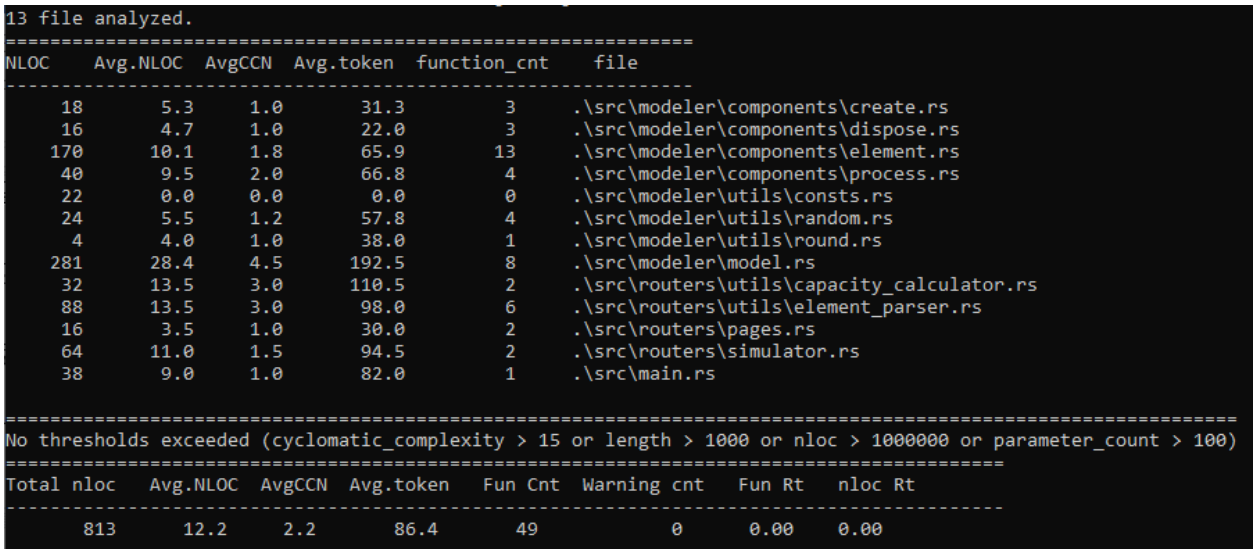
На рисунках 3.13 та 3.14 представлено фрагменти звіту для Python – та Rust-версій відповідно.



```
15 file analyzed.
==============================================================
NLOC    Avg.NLOC  AvgCCN  Avg.token  function_cnt    file
--------------------------------------------------------------
     27       7.7     1.3       42.7         3     .\src\modeler\components\create.py
     15       3.0     1.2       20.5         4     .\src\modeler\components\dispose.py
     64       4.5     1.0       29.6        13     .\src\modeler\components\element.py
     54      10.0     2.0       60.0         5     .\src\modeler\components\process.py
     10       0.0     0.0        0.0         0     .\src\modeler\utils\consts.py
     28      24.0     4.0      149.0         1     .\src\modeler\utils\shortcuts.py
    134      17.9     4.0      126.4         7     .\src\modeler\model.py
     95      17.8     4.2      116.8         5     .\src\routers\utils\element_parser.py
     17      14.0     1.0      115.0         1     .\src\routers\utils\load_calculator.py
     24      23.0     3.0      187.0         1     .\src\routers\utils\result_decoder.py
     18       0.0     0.0        0.0         0     .\src\routers\manual.py
     12       2.5     1.0       30.0         2     .\src\routers\pages.py
     32      12.0     1.0       77.5         2     .\src\routers\simulator.py
     23       5.0     2.0       33.5         4     .\src\utils\rng.py
      8       0.0     0.0        0.0         0     .\src\main.py

=================================================================================================================
No thresholds exceeded (cyclomatic_complexity > 15 or length > 1000 or nloc > 1000000 or parameter_count > 100)
=================================================================================
Total nloc   Avg.NLOC  AvgCCN  Avg.token   Fun Cnt  Warning cnt   Fun Rt   nloc Rt
---------------------------------------------------------------------------------
       561       9.7     2.1       65.9        48            0     0.00    0.00
```

Рис. 3.13. Звіт Lizard по проєкту на Python



```
13 file analyzed.
==============================================================
NLOC    Avg.NLOC  AvgCCN  Avg.token  function_cnt    file
--------------------------------------------------------------
     18       5.3     1.0       31.3         3     .\src\modeler\components\create.rs
     16       4.7     1.0       22.0         3     .\src\modeler\components\dispose.rs
    170      10.1     1.8       65.9        13     .\src\modeler\components\element.rs
     40       9.5     2.0       66.8         4     .\src\modeler\components\process.rs
     22       0.0     0.0        0.0         0     .\src\modeler\utils\consts.rs
     24       5.5     1.2       57.8         4     .\src\modeler\utils\random.rs
      4       4.0     1.0       38.0         1     .\src\modeler\utils\round.rs
    281      28.4     4.5      192.5         8     .\src\modeler\model.rs
     32      13.5     3.0      110.5         2     .\src\routers\utils\capacity_calculator.rs
     88      13.5     3.0       98.0         6     .\src\routers\utils\element_parser.rs
     16       3.5     1.0       30.0         2     .\src\routers\pages.rs
     64      11.0     1.5       94.5         2     .\src\routers\simulator.rs
     38       9.0     1.0       82.0         1     .\src\main.rs

=================================================================================================================
No thresholds exceeded (cyclomatic_complexity > 15 or length > 1000 or nloc > 1000000 or parameter_count > 100)
=================================================================================
Total nloc   Avg.NLOC  AvgCCN  Avg.token   Fun Cnt  Warning cnt   Fun Rt   nloc Rt
---------------------------------------------------------------------------------
       813      12.2     2.2       86.4        49            0     0.00    0.00
```

Рис. 3.14. Звіт Lizard по проєкту на Rust

Аналіз показує, що реалізація на Rust характеризується дещо більшим загальним обсягом коду, що є очікуваним наслідком суворішої типізації та необхідності явного опису структур і життєвих циклів даних. Водночас цикломатична складність обох реалізацій практично однакова, що свідчить про збереження логічної простоти алгоритмів і відсутність надмірного ускладнення після перепису.

Збільшення середньої кількості токенів на функцію у Rust (на ~31%) пояснюється більш докладним управлінням ресурсами та структурними деклараціями. Жодна з реалізацій не перевищила граничних значень за

критеріями складності або довжини функцій, що підтверджує високу підтримуваність і чистоту коду в обох версіях.

3.4 Висновки до розділу 3

У результаті проведеної модернізації застосунку для моделювання СМО було повністю оновлено серверну частину, реалізовану на мові Rust, при збереженні загальної логічної структури початкової системи. Архітектурні зміни зосереджувались переважно навколо переходу від наслідування до композиції, чіткішої типізації структур та оптимізації обміну даними між клієнтською та серверною частинами. Це забезпечило покращення стабільності, передбачуваності поведінки та підвищення ефективності управління пам'яттю.

Клієнтська частина зберегла інтуїтивно зрозумілий графічний інтерфейс із можливістю побудови моделей у вигляді графів, проте зазнала низки локальних оптимізацій із мінімізацією обсягу переданих на сервер даних. Компоненти формування, симуляції та виводу результатів були перероблені з урахуванням нової структури запитів та суворішої архітектури взаємодії між елементами моделі.

Проведене тестування ефективності підтвердило, що перехід на Rust забезпечив суттєве зростання продуктивності:

- час виконання симуляцій скоротився у середньому в 3 рази;
- використання оперативної пам'яті зменшилось приблизно у 2.5-3 рази.

Такі результати пояснюються низкою факторів: відсутністю інтерпретатора, компіляцією у машинний код, відсутністю Garbage Collector та ефективнішою роботою з пам’яттю. Це підтверджує доцільність вибору Rust для розв’язання задач, що вимагають стабільної продуктивності та високої ефективності використання ресурсів.

Порівняльний аналіз якості коду за допомогою інструменту Lizard показав, що хоча Rust-реалізація має більший обсяг коду, цикломатична складність залишилася практично незмінною. Це свідчить про те, що підвищення продуктивності було досягнуто не жертвуючи ускладненням логіки. Навпаки, код став більш формалізованим і структурно цілісним, а сувора типізація та композиційний підхід сприяли підвищенню надійності системи та полегшенню її підтримки.

Отже, оновлений застосунок демонструє комплексне покращення – як у швидкодії та ефективності використання ресурсів, так і у якості архітектури та структурної організації коду. Перепис серверної частини з Python на Rust виявився доцільним рішенням, що не лише збільшило продуктивність, а й підвищило технологічну зрілість проєкту, створивши надійну основу для його подальшого розвитку. Крім того, отримані результати підкреслюють важливість поєднання сучасних мов програмування та оптимізованих алгоритмів для створення високопродуктивних та надійних систем моделювання.

РОЗДІЛ 4
РОЗРОБКА СТАРТАП-ПРОЄКТУ

4.1 Загальна характеристика стартап-проєкту

Розроблений стартап-проєкт передбачає створення вебзастосунку для проєктування та аналізу систем масового обслуговування (СМО). Такий інструмент надає можливість користувачам – інженерам, аналітикам, дослідникам та студентам – швидко та легко створювати моделі черг, потоків запитів і каналів обслуговування, проводити імітаційні експерименти, аналізувати результати та виявляти «вузькі місця» у процесах.

Застосунок поєднує простоту графічного інтерфейсу з високою обчислювальною ефективністю серверної частини, реалізованої на мові Rust.

Таблиця 4.1

Опис ідеї стартап-проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Вебзастосунок для моделювання систем масового обслуговування	1. Освітня діяльність (університети, курси, навчальні центри)	Можливість швидко створювати моделі та отримувати наочні результати без глибокого знання програмування
	2. Бізнес-аналітика, оптимізація процесів обслуговування клієнтів	Підвищення ефективності бізнес-процесів за рахунок моделювання навантаження та черг
	3. Наукові дослідження у галузі моделювання та системного аналізу	Зменшення часу на проведення експериментів, можливість перевірки теоретичних результатів на практиці

Для оцінки конкурентоспроможності проведено порівняння з трьома аналогічними рішеннями:

- AnyLogic – комерційне ПЗ для моделювання систем різного типу;
- SimPy – бібліотека на Python для дискретно-подієвого моделювання;
- Arena Simulation – популярний інструмент для промислового моделювання.

Таблиця 4.2

Визначення сильних, слабких та нейтральних характеристик ідеї проєкту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проєкт	AnyLogic	SimPy	Arena			
1	Продуктивність виконання моделей	Висока (Rust)	Середня	Низька	Середня			+
2	Зручність користування (UI/UX)	Висока	Середня	Низька	Середня			+
3	Вартість використання	Безкоштовно	Платна ліцензія	Безкоштовно	Платна		+	
4	Розширюваність та модульність	Висока	Висока	Середня	Низька			+

Таблиця 4.2 (продовження)

Визначення сильних, слабких та нейтральних характеристик ідеї проєкту

5	Мова реалізації та швидкодія	Rust (компі-льована)	Java	Python	C/C++			+
6	Можливості інтеграції з іншими системами	Високі (REST API)	Високі	Середні	Низькі			+
7	Підтримка документації та спільноти	Середня	Висока	Висока	Середня	+		
8	Вимоги до ресурсів	Низькі	Середні	Низькі	Високі			+
9	Легкість розгортання	Висока (веб-доступ)	Середня	Складна (Python-середо-вище)	Середня			+

4.2 Технологічний аудит проєкту

Розробка вебзастосунку для проєктування та аналізу систем масового обслуговування потребує аналізу наявних аналогів та проблем з якими можна зіткнутися, у випадку відсутності засобів.

Таблиця 4.3

Технологічна здійсненність ідеї проєкту

№ п/п	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Реалізація компонента графічного конструктора	Відкриті бібліотеки та модулі, приклади реалізації	Необхідно розробити	Засоби є загальнодоступними
2	Реалізація компонента формування графу систем масового обслуговування	Існуючі приклади реалізації підходу	Наявна	Засоби є загальнодоступними
3	Реалізація компонента симуляції систем масового обслуговування	Існуючі приклади реалізації підходу	Наявна	Засоби є загальнодоступними
4	Реалізація компонента виводу результатів роботи системи масового обслуговування	Відкриті бібліотеки та модулі, приклади реалізації	Необхідно розробити	Засоби є загальнодоступними
5	Програмний продукт, що включає всі вказані компоненти	Програмні засоби для розробки модуля та описані вище підходи	Необхідно розробити	Засоби є загальнодоступними

Таблиця 4.3 (продовження)

Технологічна здійсненність ідеї проєкту

Обрані технології реалізації ідеї проєкту: Rust, Rocket, cpu-time, sysinfo, JavaScript, drawflow, chart.js, bootstrap

Можна зробити висновок, що реалізація цієї ідеї можливо і включає в себе розробку нових модулів та використання відкритих бібліотек.

4.3 Аналіз ринкових можливостей запуску стартап-проєкту

Попит на інструменти моделювання та аналізу СМО стабільно зростає завдяки розвитку IT-сфери, автоматизації бізнес-процесів, а також підвищенню інтересу до аналітичних і симуляційних систем. Вебзастосунки такого типу застосовуються у наукових, інженерних, транспортних, фінансових і телекомунікаційних галузях, де потрібно оцінювати навантаження на сервери, мережі, виробничі або логістичні процеси.

Таблиця 4.4

Попередня характеристика потенційного ринку стартап-проєкту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	4
2	Загальний обсяг продаж, грн/ум.од	-
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Високі вимоги до надійності системи, необхідність технічної експертизи у моделюванні та статистиці

Можна зробити висновок, що ринок є привабливим для входження за попереднім оцінюванням.

Таблиця 4.4 (продовження)

Попередня характеристика потенційного ринку стартап-проєкту

5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі (або по ринку), %	25%

У таблиці 4.5 наводяться визначені потенційні клієнти, а також їх характеристика.

Таблиця 4.5

Характеристика потенційних клієнтів стартап-проєкту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Необхідність аналізу ефективності систем обслуговування	Науковці, викладачі, студенти технічних спеціальностей	Зосереджені на точності, достовірності результатів, простоті у використанні; орієнтація на навчальні та дослідницькі задачі	– Інтуїтивний інтерфейс; – можливість швидкої побудови моделей; – відкритість вихідних даних; – безкоштовний доступ або академічна ліцензія.

Таблиця 4.5 (продовження)

Характеристика потенційних клієнтів стартап-проєкту

2	Потреба в оптимізації бізнес-процесів	Аналітики підприємств, менеджери з операційної ефективності	Орієнтовані на результативність і практичну користь; очікують автоматизації звітності	– Інтеграція з базами даних; – можливість експорту звітів; – висока швидкість симуляції.
3	Необхідність перевірки інженерних рішень до впровадження	Інженери, системні архітектори	Потребують точності моделювання, можливості налаштування параметрів, гнучкої архітектури	– Підтримка складних сценаріїв; – масштабованість моделей; – надійність обчислень.
4	Використання у навчальних програмах та онлайн-курсах	Освітні платформи, університети	Високий попит на вебінтерфейс та простоту взаємодії, необхідність багаторазового доступу	– Хмарний формат роботи; – простота доступу через браузер; – можливість спільної роботи.

Для виявлення можливих загроз, які можуть вплинути на успішність виходу стартап-проєкту на ринок, доцільно проаналізувати зовнішні та внутрішні фактори. У таблиці 4.6 наведено основні фактори загроз, їхній зміст та можливі реакції команди проєкту на їх усунення або мінімізацію.

Таблиця 4.6

Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Поява нових конкурентів з аналогічним функціоналом	Розвиток нових або оновлених рішень з моделювання СМО (наприклад, закордонні SaaS-сервіси) може ускладнити залучення користувачів	Активне позиціонування на академічному та освітньому ринках, впровадження унікальних функцій симуляції, інтеграція з навчальними платформами
2	Нестабільність економічного середовища	Зниження інвестиційної активності або платоспроможності користувачів може вплинути на розвиток і підтримку продукту	Розробка безкоштовної базової версії, залучення грантового фінансування, орієнтація на міжнародні ринки
3	Відсутність довіри з боку користувачів до нового продукту	Потенційні користувачі можуть сумніватися у точності моделей або стабільності сервісу	Демонстрація кейсів використання, публікація результатів тестувань, створення відкритої документації та користувацької підтримки
4	Технологічні обмеження серверної інфраструктури	Велика кількість користувачів або складні симуляції можуть перевищувати потужності системи	Модернізація серверної частини, перехід на хмарні обчислення (AWS, GCP), використання розподілених систем

Таблиця 4.6 (продовження)

Фактори загроз

5	Витіснення з боку великих IT-компаній	Корпорації можуть запустити власні рішення з подібною функціональністю	Формування партнерств з університетами та науковими установами, орієнтація на нішевий сегмент (освітній, дослідницький)

У таблиці 4.7 наведені фактори можливостей, що сприяють ринковому впровадженню.

Таблиця 4.7

Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція команії
1	Зростання попиту на онлайн-освітні сервіси	Висока популярність цифрових інструментів навчання та тренажерів з моделювання процесів, зокрема серед студентів технічних спеціальностей	Розробка інтеграцій із системами дистанційного навчання (Moodle, Google Classroom), створення навчальних кейсів
2	Можливість виходу на міжнародний ринок	Вебзастосунок не має мовних або географічних обмежень, тому може бути адаптований для навчальних закладів за кордоном	Локалізація інтерфейсу (французька, польська, німецька), публікація продукту на міжнародних освітніх платформах

Таблиця 4.7 (продовження)

Фактори можливостей

3	Підтримка з боку університетів та наукових спільнот	Попит на зручні інструменти моделювання СМО у студентів, викладачів і дослідників	Укладання партнерських угод з університетами, впровадження безкоштовних академічних ліцензій
4	Розвиток хмарних технологій	Використання хмарних обчислень дозволяє підвищити масштабованість, доступність і швидкість роботи системи	Міграція серверної частини до хмарної інфраструктури, забезпечення високої продуктивності при мінімальних витратах
5	Додаткові можливості монетизації	Можливість створення платних модулів для розширеної аналітики або інтеграцій із корпоративними системами	Впровадження гібридної моделі монетизації (freemium), пропозиція кастомних рішень для бізнесу та освіти

У таблиці 4.8 наведено загальні риси конкуренції на ринку.

Таблиця 4.8

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Тип конкуренції: монополістична	На ринку присутні кілька великих міжнародних гравців, але продукт кожного має власну спеціалізацію та унікальні риси	Створення власної ринкової ніші завдяки поєднанню доступності, локалізації та зручного інтерфейсу
2. За рівнем конкурентної боротьби: глобальний	Конкуренція формується на міжнародному рівні – більшість подібних платформ орієнтовані на світовий ринок	Виведення українського продукту на глобальний рівень через відкритий доступ
3. За галузевою ознакою: внутрішньогалузева	Конкуренція ведеться всередині галузі розробки програмного забезпечення для моделювання та симуляції процесів	Забезпечення унікальності продукту завдяки фокусу на системах масового обслуговування та імітаційному підході
4. Конкуренція за видами товарів: товарно-видова	Продукти мають спільну мету – симуляцію процесів, але різняться підходами, технологіями та глибиною моделювання	Покращення точності та швидкодії симуляцій, додавання інструментів аналізу результатів, підтримка різних моделей СМО

Таблиця 4.8 (продовження)

Ступеневий аналіз конкуренції на ринку

5. За характером конкурентних переваг: нецінова	Основна конкуренція відбувається не за ціною, а за функціональністю, продуктивністю та якістю користувацького досвіду	Оптимізація алгоритмів, підвищення стабільності, поліпшення UX/UI, впровадження інтерактивних графічних інтерфейсів
6. За інтенсивністю: не марочна	Сильний бренд у цій ніші мають лише окремі міжнародні компанії, проте більшість користувачів обирають за якістю продукту	Формування власного бренду через наукову спільноту, освітні партнерства, участь у хакатонах та конференціях

У таблиці 4.9 наведено більш детальні риси конкуренції на ринку за М. Портером.

Таблиця 4.9

Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Складові аналізу	AnyLogic, Simul8, Arena Simulation, SimPy, NetLogo, GPSS World	Нові хмарні сервіси	Постачальники хмарних серверів (AWS, Azure, GCP) та бібліотек з відкритим кодом; сила постачальників середня	Академічні заклади, дослідницькі інститути, компанії у сфері логістики, IT; мають помірний вплив	Табличні моделі, статистичні пакети (Excel, R, Python pandas); можуть частково замінювати симуляційні рішення
Висновки	Інтенсивність конкуренції середня: міжнародні продукти є, але ринок в Україні малозаповнений	Існують потенційні конкуренти, однак високі технічні бар'єри входу стримують нових гравців	Постачальники не диктують умов – більшість ресурсів доступні у відкритому доступі	Клієнти частково можуть диктувати вимоги (зручність, локалізація, точність моделі)	Ризик появи замінників середній – можуть використовуватись альтернативні аналітичні системи, але вони не забезпечують моделювання в динаміці

На основі вище наведених таблиць виведено фактори конкурентоспроможності, та наведено у таблиці 4.10.

Таблиця 4.10

Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Функціональна повнота	Вебзастосунок включає модулі побудови графічних схем, формування графів, симуляції та виводу результатів. Це дозволяє виконувати повний цикл аналізу СМО без потреби у сторонніх інструментах.
2	Інтерактивність та зручність інтерфейсу	Використання сучасних фреймворків (React, D3.js) забезпечує інтуїтивну візуалізацію процесів і зручну взаємодію користувача з моделлю. Це підвищує залучення користувачів і спрощує навчання.
3	Відкритість і масштабованість технологій	Розробка базується на відкритому програмному забезпеченні (JavaScript, Rust), що дозволяє легко розширювати функціонал і уникати високих витрат на ліцензії.
4	Хмарна доступність і кросплатформність	Вебархітектура дає змогу працювати з будь-якого пристрою без встановлення ПЗ, що особливо актуально для освітніх закладів і командних проєктів.
5	Гнучкість налаштування системи	Можливість зміни параметрів черг, інтенсивності потоків, кількості серверів тощо робить систему універсальною для різних сценаріїв.

Таблиця 4.10 (продовження)

Обґрунтування факторів конкурентоспроможності

6	Цінова доступність	У порівнянні з комерційними аналогами (Arena, AnyLogic), застосунок може бути запропонований за нижчою ціною або у freemium-форматі.
7	Підтримка користувачів та документація	Передбачено інтеграцію довідкових матеріалів, навчальних прикладів і онлайн-підтримки, що полегшує впровадження продукту.

Порівняльний аналіз сильних та слабких сторін системи збору та агрегації даних для аналізу вакансій відображено у таблиці 4.11.

Таблиця 4.11

Порівняльний аналіз сильних та слабких сторін вебзастосунку для проєктування та аналізу систем масового обслуговування

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з вебзастосунком для проєктування та аналізу систем масового обслуговування						
			-3	-2	-1	0	+1	+2	+3
1	Зручність інтерфейсу та простота використання	15					+		
2	Функціональні можливості моделювання	17				0			
3	Вартість використання	20		+					
4	Інноваційність	17			+				

Враховуючи вище наведені таблиці зробимо висновок ринкового аналізу, та сформуємо його у вигляді SWOT- аналізу (таблиця 4.12).

Таблиця 4.12

SWOT- аналіз стартап-проекту

Сильні сторони:	Слабкі сторони:
– Комплексне рішення для побудови, симуляції та аналізу СМО в одному середовищі; – використання відкритих технологій (JavaScript, Python, Node.js); – простий інтерфейс, придатний для навчальних цілей; – хмарна доступність і кросплатформність; – гнучкість налаштування моделей під різні сценарії.	– Недостатня впізнаваність бренду на ринку; – обмежені ресурси для масштабування; – невеликий маркетинговий бюджет; – відсутність офлайн-режиму; – немає інтеграції з корпоративними платформами.
Можливості:	Загрози:
– Зростання попиту на освітні моделювальні інструменти; – вихід на міжнародний EdTech-ринок; – інтеграція з навчальними платформами (Moodle, Google Classroom); – підтримка відкритого коду та залучення IT-спільноти; – можливість грантового фінансування.	– Поява нових сильних конкурентів; – економічна нестабільність і скорочення фінансування освіти; – недовіра до маловідомих рішень; – технічні ризики (сервери, хмари); – імітація функцій конкурентами.

На основі SWOT-аналізу визначено альтернативи ринкового впровадження стартап-проекту (таблиця 4.13).

Таблиця 4.13

Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Розгортання MVP-версії вебзастосунку з базовими функціями моделювання та симуляції СМО; залучення користувачів із навчальних закладів і наукових спільнот.	Висока	3-6 місяців
2	Запуск пілотної програми співпраці з університетами та IT-компаніями для тестування та спільної апробації системи.	Середня	6-9 місяців
3	Проведення маркетингової кампанії в соціальних мережах і спеціалізованих форумах, спрямованої на популяризацію продукту серед студентів і дослідників.	Висока	2-4 місяці
4	Розширення функціоналу системи (інтеграція з навчальними платформами, API, графічні інтерфейси аналітики) після залучення інвестицій.	Середня	9-12 місяців
5	Вихід на міжнародний ринок EdTech-платформ шляхом адаптації інтерфейсу та локалізації системи.	Низька	12-18 місяців

Аналіз альтернатив показує, що найбільш доцільним є етапне впровадження продукту – спочатку запуск мінімально життєздатної версії (MVP) з подальшим розширенням функціоналу після отримання користувацького фідбеку та додаткових ресурсів. Такий підхід мінімізує ризики та забезпечить гнучке реагування на зміни ринкового середовища.

4.4 Розробка ринкової стратегії проєкту

Визначення цільових груп потенційних клієнтів базується на результатах попереднього маркетингового аналізу, який показав наявність попиту з боку освітніх, наукових та комерційних структур, зацікавлених у моделюванні процесів обслуговування та оптимізації систем (таблиця 4.14).

Таблиця 4.14

Вибір цількових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Викладачі та студенти технічних університетів, що досліджують системи масового обслуговування	Висока	Стабільний і зростаючий	Низька	Висока

Таблиця 4.14 (продовження)

Вибір цількових груп потенційних споживачів

2	Наукові установи, що проводять дослідження у галузі моделювання, оптимізації та теорії черг	Середня	Помірний	Середня	Середня
3	IT-компанії, що займаються розробкою або тестуванням моделей оптимізації бізнес-процесів	Висока	Зростаючий	Висока	Середня
4	Освітні онлайн-платформи, зацікавлені в інтеграції симуляційних інструментів	Середня	Помірний	Середня	Середня

Таблиця 4.14 (продовження)

Вибір цількових груп потенційних споживачів

5	Державні та приватні організації, що аналізують навантаження на інфраструктурні системи (транспорт, зв'язок, медицина)	Низька	Обмежений	Висока	Низька
Які цільові групи обрано: викладачі і студенти технічних університетів, ІТ-компанії, наукові установи.					

При виборі одного сегменту використаємо стратегію диференційованого маркетингу та визначимо її базову стратегію розвитку (таблиця 4.15).

Таблиця 4.15

Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Розроблення вебзастосунку для моделювання СМО та його впровадження у навчальних, наукових і комерційних цілях	Диференційований маркетинг	– Унікальний функціонал моделювання без необхідності локальної інсталяції; – доступність із будь-якого пристрою; – інтерактивні графіки та зручний інтерфейс; – можливість адаптації під конкретні задачі користувачів.	Стратегія диференціації

Таблиця 4.15 (продовження)

Визначення базової стратегії розвитку

2	Співпраця з навчальними платформами та IT-компаніями для інтеграції сервісу у їхні екосистеми	Концентрований маркетинг	– Гнучкість у налаштуванні; – API та плагіни для інтеграції; – висока стабільність і безпека роботи.	Стратегія спеціалізації

У таблиці 4.16 описано стратегію базової конкурентної поведінки.

Таблиця 4.16

Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
1	Так, на українському ринку аналогів серед онлайн-платформ для моделювання СМО практично немає	Переважно пошук нових споживачів – студентів, науковців, викладачів, IT-фахівців	Частково може запозичувати окремі зручні елементи інтерфейсу або принципи візуалізації даних із зарубіжних аналогів	Інноваційна стратегія

Таблиця 4.16 (продовження)

Визначення базової стратегії конкурентної поведінки

2	Ні, на міжнародному ринку існують подібні рішення	Компанія частково орієнтується на залучення користувачів конкурентів завдяки кращій доступності та простоті використання	Може наслідувати загальні принципи побудови інтерфейсу конкурентів для забезпечення зручності користувачів	Імітаційно-адаптаційна стратегія
3	Так, у сегменті навчального використання	Пошук нових споживачів серед закладів освіти, які ще не мають цифрових лабораторій СМО	Копіювання не планується, створюються оригінальні навчальні сценарії	Стратегія диференціації

Враховуючи описані вище стратегії визначимо стратегію позиціонування продукту (таблиця 4.17).

Таблиця 4.17

Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту
1	Простота використання без складного навчання	Стратегія диференціації	Інтуїтивно зрозумілий інтерфейс, швидкий старт роботи, сучасний UX	Простота, інтелектуальність, доступність
2	Доступна ціна при професійному рівні можливостей	Стратегія диференціації	Оптимальне співвідношення вартості та функціоналу, фріміум-модель	
3	Інноваційність, веб-доступність, інтеграція з навчальними середовищами	Стратегія диференціації	Онлайн-платформа, інтеграція з освітніми інструментами, регулярні оновлення	

4.5 Розроблення маркетингової програми стартап-проєкту

Для формування маркетингової концепції продукту, спершу треба визначити результати аналізу конкурентоспроможності продукту (таблиця 4.18).

Таблиця 4.18

Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Необхідність швидко створювати моделі систем масового обслуговування (черги, потоки заявок тощо)	Простий графічний конструктор моделі у браузері без потреби в інсталяції	Інтерактивний вебінтерфейс; не потребує встановлення; робота напряму у браузері
2	Бажання скоротити час на симуляцію моделей	Висока швидкодія симуляції завдяки використанню мови Rust	Продуктивність у 3 рази вища за аналоги на Python або JavaScript; низьке споживання ресурсів
3	Потреба у точному аналізі продуктивності системи	Детальна статистика та графіки результатів (ймовірність втрат, довжина черги, простої тощо)	Повна візуалізація; автоматичне формування звітів; зручний інтерфейс для порівняння
4	Потреба у гнучкому налаштуванні параметрів моделі	Можливість задавати будь-які параметри для кожного елементу (інтенсивність потоків, кількість каналів і т.д.)	Гнучкість у побудові моделей будь-якої складності

Таблиця 4.18 (продовження)

Визначення ключових переваг концепції потенційного товару

5	Необхідність інтеграції в навчальний процес або наукові дослідження	Можливість використання застосунку як навчального або дослідницького інструменту	Відкритість архітектури, модульність, локальне збереження моделей, зрозумілий UI
6	Потреба у прогнозуванні «вузьких місць» системи	Автоматичне визначення потенційних перевантажень перед запуском симуляції	Унікальний алгоритм прогнозування вузьких місць

Певну популярність отримала модель маркетингу, що складається з трьох рівнів:

1) споживачі;
2) інструменти маркетингу;
3) політика комунікацій.

У таблиці 4.19 наведена трирівнева маркетингова модель товару.

Таблиця 4.19

Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
I. Товар за задумом	Вебзастосунок для проєктування та аналізу систем масового обслуговування

Таблиця 4.19 (продовження)

Опис трьох рівнів моделі товару

II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1) Швидкодія 2) Ефективність 3) Користувацький інтерфейс	1) Нм 2) Нм 3) Нм	1) Тх/Тл/Е 2) Тх/Тл 3) Е
	Якість: стабільна робота в браузері, перевірені алгоритми моделювання, регулярні оновлення		
	Пакування: власний сайт		
	Марка: Model’s Construct		
III. Товар із підкріпленням	Програмний продукт		
	Програмний продукт		
За рахунок чого потенційний товар буде захищено від копіювання: захищений програмний код, а також використання ліцензії.			

Вартість на рекламу у товарі визначається типом реклами кількістю її відображення. У таблиці 4.20 наведено ціни відповідно до кількості реклами.

Таблиця 4.20

Визначення меж встановлення ціни

№ п/п	Рівень цін на товари замінники, тис. грн/програмний продукт	Рівень цін на товари аналоги, тис. грн/програмний продукт	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	-	Від 400 до 2500 USD за ліцензію	Середній-високий	Нижня межа – 0 грн (безкоштовна версія або демо); верхня межа – 6000-8000 грн/міс (для професійного доступу або корпоративної ліцензії).

У таблиці 4.21 наведена оптимальна система збуту.

Таблиця 4.21

Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Оплата доступу до вебзастосунку дослідниками, студентами та викладачами, які використовують моделювання СМО у своїй роботі або навчанні. Здійснюється онлайн через офіційний сайт	Продаж ліцензій та підписок; технічна підтримка користувачів; оновлення програмного забезпечення; забезпечення роботи платформи	Прямий збут через вебплатформу	Власна система збуту через офіційний вебсайт із можливістю онлайн-оплати, автоматичною активацією доступу та просуванням через наукові спільноти й освітні заклади

У таблиці 4.22 наведено концепцію маркетингових комунікацій.

Таблиця 4.22

Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Викладачі, студенти, аналітики та дослідники, які працюють із моделями черг, навантаженням систем і симуляціями для наукових або навчальних цілей	Вебсайти університетів, освітні платформи, наукові форуми, LinkedIn, YouTube, тематичні блоги, Telegram-канали для IT і освіти	Зручність створення і тестування моделей СМО без складного програмування; можливість швидко аналізувати результати	Показати науковцям і студентам, що «Model's Construct» економить час і спрощує навчання теорії масового обслуговування, забезпечуючи професійний рівень результатів	Моделювання систем масового обслуговування без складного коду

4.6 Висновки до розділу 4

У четвертому розділі дипломної роботи була визначена актуальність стартапу «Model’s Construct», проаналізовано конкурентне середовище та підтверджено технологічну здійсненність проєкту.

Було проведено аналіз ринкових можливостей для запуску продукту та розроблено маркетингову стратегію просування.

Незважаючи на наявність конкурентів на ринку систем моделювання, вони не поєднують усі ті функціональні можливості, які пропонує розроблений застосунок. Основними характеристиками конкурентоспроможності продукту є простота використання, висока швидкодія завдяки мові програмування Rust, універсальність та орієнтованість на освітній і дослідницький сегменти.

Імплементація продукту є доцільною, оскільки доведено його технологічну здійсненність, а також ринкову перспективність у сфері аналітичних вебзастосунків.

Отже, можна зробити висновок, що стартап «Model’s Construct» має високий потенціал успіху за умови реалізації продукту мінімальної вартості з основними функціями та поступового розширення його можливостей після виходу на ринок.

ВИСНОВКИ

Магістерська робота «Вебзастосунок для проєктування та аналізу систем масового обслуговування» присвячена розробці та вдосконаленню інструменту, який дозволяє ефективно будувати, аналізувати та оптимізувати моделі СМО у вебсередовищі. Актуальність теми зумовлена зростаючою потребою у гнучких та продуктивних засобах моделювання процесів обслуговування, які знаходять широке застосування у телекомунікаційних, транспортних, виробничих, освітніх та аналітичних системах.

У першому розділі було проведено теоретичний аналіз предметної області систем масового обслуговування. Визначено основні типи, характеристики та методи моделювання СМО, зокрема аналітичні та імітаційні підходи. Результати аналізу стали науково-методичною основою для розробки вебзастосунку, що дозволяє не лише відтворювати теоретичні моделі, а й застосовувати їх до практичних задач. Визначено ключові фактори ефективності СМО та обмеження класичних методів, що дало змогу сформувати вимоги до майбутньої системи.

У другому розділі проведено аналіз технологічної бази існуючого застосунку та оцінено можливості його оптимізації. Виявлено, що початкова реалізація на мові Python, хоч і забезпечувала швидку розробку, мала суттєві обмеження у продуктивності. Проведене дослідження показало, що для підвищення ефективності системи доцільним є перехід на сучасну мову Rust, яка поєднує високу швидкодію з безпечністю роботи з пам’яттю. Вибір фреймворку Rocket для серверної частини виявився оптимальним рішенням для забезпечення стабільності, гнучкості та продуктивності вебзастосунку. Таким чином, було обґрунтовано технологічний перехід, який став основою подальшої модернізації.

У третьому розділі безпосередньо реалізовано оновлення серверної частини на Rust із збереженням логічної структури початкової системи. Проведено архітектурну реструктуризацію з переходом до композиційного

підходу, удосконалено типізацію структур та обмін даними між клієнтською та серверною частинами. Тестування показало, що модернізована версія продемонструвала значне зростання ефективності:

- час симуляції скоротився у 3 рази,
- використання оперативної пам’яті зменшилося у 2.5–3 рази.

При цьому складність логіки залишилась незмінною, а якість коду зросла завдяки суворій типізації та більшій формальності архітектури. Це доводить, що перехід на Rust є доцільним рішенням для проєктів, де критичними є швидкодія, стабільність і безпечність виконання.

У четвертому розділі було розглянуто питання комерціалізації продукту та побудови маркетингової стратегії для стартапу «Model’s Construct». Проведено аналіз конкурентного середовища та визначено переваги розробленого застосунку – простоту використання, високу продуктивність, універсальність і можливість застосування як у навчальних, так і у дослідницьких цілях. Підтверджено технологічну здійсненність та ринкову доцільність продукту, визначено його позиціонування на ринку як гнучкого інструменту для моделювання та аналізу систем.

Отже, у результаті виконання магістерської роботи:

- створено науково обґрунтовану модель вебзастосунку для моделювання СМО;
- проведено оптимізацію архітектури та технологічного стеку;
- підтверджено ефективність переходу на Rust;
- розроблено маркетингову концепцію стартапу та обґрунтовано його ринкову перспективність.

Розроблений застосунок «Model’s Construct» поєднує наукову коректність моделювання з високою швидкодією та практичною зручністю. Його реалізація робить вагомий внесок у розвиток інструментів цифрового моделювання та створює підґрунтя для подальших досліджень у галузі аналітичних вебсистем і оптимізації процесів обслуговування. Він має потенціал стати ефективним інструментом для дослідження, викладання та

проектування систем масового обслуговування, а також успішним комерційним продуктом у сфері аналітичних вебсервісів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Green L. Queueing Theory and Modeling. *Columbia Business School*. URL: https://business.columbia.edu/sites/default/files-efs/pubfiles/5474/queueing%20theory%20and%20modeling.pdf.
2) Constantin H. Markov Chains and Queueing Theory. *University of Chicag*o. 2011. URL: https://www.math.uchicago.edu/~may/VIGRE/VIGRE2011/REUPapers/Constantin.pdf.
3) Güneş M. Queueing Models. *Freie Universität Berlin*. URL: https://www.mi.fu-berlin.de/inf/groups/ag-tech/intern/19540-V-Simulation/08_Queueing_Models.pdf.
4) Ledder G. Introduction to Queueing Theory: A Modeling Perspective. *University of Nebraska-Lincoln*. 2019. URL: https://www.math.unl.edu/~gledder1/Notes/428/Queueing%20Theory%20Notes.pdf.
5) Riska A., Smirni E. MAMSolver: A matrix analytic methods tool. *College of William and Marry Williamsburg*. URL: https://www.cs.wm.edu/MAM-Solver/paper-mamsolve.pdf.
6) Неруш В. Б., Курдеча В. В. Імітаційне моделювання систем та процесів. *Національний технічний університет України "Київський політехнічний інститут"*. URL: https://ela.kpi.ua/server/api/core/bitstreams/3fe27852-776f-4130-bc62-c9c2be44ed6e/content.
7) Desai J. Performance Comparison of Python, Golang, Rust and C++. *Jinal Desai*. URL: https://jinaldesai.com/performance-comparison-of-python-go-lang-rust-and-c/.
8) Campbell R. H., Wong R. S. K. A Survey of the Literature on Dynamic Programming. *Proceedings of the 1976 ACM Annual Conference*. 1976. P. 176–180. URL: https://doi.org/10.5555/957289.957341.

9) Ian Z. R. Rust vs. C++: a Modern Take on Performance and Safety. *The New Stack*. URL: https://thenewstack.io/rust-vs-c-a-modern-take-on-performance-and-safety/.
10) Петренко В. Companies That Use Rust Language: Real-World Examples from Leading Businesses. Litslink. URL: https://litslink.com/blog/companies-that-use-rust-language.
11) KRust: A Formal Executable Semantics of Rust / F. Wang et al. *arxiv*. 2018. URL: https://doi.org/10.48550/arXiv.1804.10806.
12) Beckley A. Measuring The Execution Times of C Versus Rust. *towards data science*. URL: https://towardsdatascience.com/measuring-the-execution-times-of-c-versus-rust-bc45c577052a/.
13) Jess A. Rust: The Perfect Language For Blockchain Development. *Imagination*. URL: https://www.itmagination.com/blog/rust-development-blockchain-web3.
14) Nam T. D. Actix (Rust). *Sharkbench*. URL: https://sharkbench.dev/web/rust-actix.
15) Difference Between User-CPU-Time and System-CPU-Time in UNIX. *GeeksForGeeks*. URL: https://www.geeksforgeeks.org/linux-unix/difference-between-user-cpu-time-and-system-cpu-time-in-unix/.
16) Cyclomatic Complexity. *GeeksForGeeks*. URL: https://www.geeksforgeeks.org/dsa/cyclomatic-complexity/.
17) Stack Overflow Annual Developer Survey. Stack Overflow. URL: https://survey.stackoverflow.co/.

ДОДАТОК А
ЛІСТИНГ ПРОГРАМИ

```
use rand::rngs::SmallRng;

use crate::modeler::components::element::Element;

pub fn in_act() {
    // nothing
}

pub fn out_act(e: &mut Element, rng: &mut SmallRng) {
    e.quantity += 1;
    e.put_tnext(e.tcurr + e.get_delay(rng));
    e.pop_tnext();
}

pub fn get_summary(e: &Element, nearest_tnext: String) -> String {
    format!(
        "\n
        ##### {} #####\n
        quantity = {} | queue = {} | tnext = {} | average_load = {:.4}\n
        failure = {}",
        e.name, e.quantity, e.queue, nearest_tnext, e.average_load, e.failure
    )
}

use rand::rngs::SmallRng;

use crate::modeler::components::element::Element;

pub fn in_act(e: &mut Element, rng: &mut SmallRng) {
    e.out_act(rng);
}

pub fn out_act(e: &mut Element) {
    e.quantity += 1;
}

pub fn get_summary(e: &Element) -> String {
    format!(
        "\n
        ##### {} #####\n
        quantity = {}",
        e.name, e.quantity
    )
}

use ordered_float::OrderedFloat;
```

```
use rand::rngs::SmallRng;
use rocket::serde::{Deserialize, Serialize};
use std::cmp::Reverse;
use std::collections::BinaryHeap;
use std::sync::atomic::{AtomicUsize, Ordering};

use crate::modeler::components::create;
use crate::modeler::components::dispose;
use crate::modeler::components::process;
use crate::modeler::utils::consts::{DistributionType, ElementType, NextEle-
mentType};
use crate::modeler::utils::random;

static NEXT_ID: AtomicUsize = AtomicUsize::new(0);

fn get_next_id() -> usize {
    NEXT_ID.fetch_add(1, Ordering::SeqCst)
}

pub fn reset_next_id() {
    NEXT_ID.store(0, Ordering::SeqCst);
}

#[derive(Clone, Serialize, Deserialize, Debug)]
pub struct Element {
    pub id: usize,
    pub name: String,
    pub worker_count: u32,
    pub elem_type: ElementType,

    #[serde(deserialize_with = "deserialize_tnext")]
    pub tnext: BinaryHeap<Reverse<OrderedFloat<f64>>>,
    pub tcurr: f64,

    pub distribution: DistributionType,
    pub delay_mean: f64,
    pub delay_dev: f64,

    pub next_element_type: NextElementType,
    pub next_elements: Vec<usize>,

    pub state: u32,
    pub queue: u32,
    pub quantity: u32,
    pub average_load: f64,

    // process-specific fields
    pub max_queue: u32,
    pub mean_queue: f64,
    pub wait_start: f64,
    pub wait_time: f64,
```

```
    pub failure: u32,
    pub state_sum: u32,
}

impl Element {
    pub fn new(
        worker_count: u32,
        delay_mean: f64,
        delay_dev: f64,
        elem_type: ElementType,
        distribution: DistributionType,
        next_element_type: NextElementType,
        max_queue: u32,
    ) -> Self {
        // prepare element data
        let id = get_next_id();
        let name;
        match elem_type {
            ElementType::Create => name = format!("Create{}", id),
            ElementType::Process => name = format!("Process{}", id),
            ElementType::Dispose => name = format!("Dispose{}", id),
        }

        let mut element = Element {
            id,
            name,
            worker_count,
            elem_type,
            tnext: BinaryHeap::new(),
            tcurr: 0.0,
            distribution,
            delay_mean,
            delay_dev,
            next_element_type,
            next_elements: Vec::new(),
            queue: 0,
            quantity: 0,
            average_load: 0.0,
            state: 0,
            max_queue,
            mean_queue: 0.0,
            wait_start: 0.0,
            wait_time: 0.0,
            failure: 0,
            state_sum: 0,
        };

        // initialize tnext based on element type
        match element.elem_type {
            ElementType::Create => element.put_tnext(0.00001),
            ElementType::Process => element.put_tnext(f64::INFINITY),
```

```
            ElementType::Dispose => element.put_tnext(f64::INFINITY),
        }

        element
    }

    pub fn get_delay(&self, rng: &mut SmallRng) -> f64 {
        match self.distribution {
            DistributionType::Exponential => random::exponential(rng, self.de-
lay_mean as f64),
            DistributionType::Normal => {
                random::normal(rng, self.delay_mean as f64, self.delay_dev as
f64)
            }
            DistributionType::Uniform => random::uniform(
                rng,
                self.delay_mean as f64 - self.delay_dev as f64,
                self.delay_mean as f64 + self.delay_dev as f64,
            ),
            DistributionType::Erlang => {
                random::erlang(rng, self.delay_mean as f64, self.delay_dev as
usize)
            }
            DistributionType::Constant => self.delay_mean,
        }
    }

    pub fn get_tnext(&self) -> f64 {
        self.tnext
            .peek()
            .map_or(f64::INFINITY, |x| x.0.into_inner())
    }

    pub fn put_tnext(&mut self, t: f64) {
        self.tnext.push(Reverse(OrderedFloat(t)));
    }

    pub fn pop_tnext(&mut self) {
        self.tnext.pop();
    }

    pub fn in_act(&mut self, rng: &mut SmallRng) {
        match self.elem_type {
            ElementType::Create => create::in_act(),
            ElementType::Process => process::in_act(self, rng),
            ElementType::Dispose => dispose::in_act(self, rng),
        }
    }

    pub fn out_act(&mut self, rng: &mut SmallRng) {
        match self.elem_type {
```

```
            ElementType::Create => create::out_act(self, rng),
            ElementType::Process => process::out_act(self, rng),
            ElementType::Dispose => dispose::out_act(self),
        }
    }

    fn get_nearest_tnext(&mut self) -> String {
        let nearest_tnext = self.get_tnext();
        if nearest_tnext != f64::INFINITY {
            self.average_load = self.quantity as f64 / nearest_tnext as f64;
            format!("{:.4}", nearest_tnext)
        } else {
            String::from("maxval")
        }
    }

    pub fn get_summary(&mut self) -> String {
        let nearest_tnext = self.get_nearest_tnext();
        match self.elem_type {
            ElementType::Create => create::get_summary(self, nearest_tnext),
            ElementType::Process => process::get_summary(self, nearest_tnext),
            ElementType::Dispose => dispose::get_summary(self),
        }
    }

    pub fn do_statistics(&mut self, delta: f64) {
        match self.elem_type {
            ElementType::Create => {}
            ElementType::Process => process::do_statistics(self, delta),
            ElementType::Dispose => {}
        }
    }
}

fn deserialize_tnext<'de, D>(
    deserializer: D,
) -> Result<BinaryHeap<Reverse<OrderedFloat<f64>>>, D::Error>
where
    D: serde::Deserializer<'de>,
{
    let raw: Vec<Option<f64>> = Vec::deserialize(deserializer)?;
    let heap = raw
        .into_iter()
        .filter_map(|opt| opt.map(|v| Reverse(OrderedFloat(v))))
        .collect();
    Ok(heap)
}

use rand::rngs::SmallRng;

use crate::modeler::components::element::Element;
```

```
pub fn in_act(e: &mut Element, rng: &mut SmallRng) {
    if e.state < e.worker_count {
        if e.state == 0 {
            e.wait_time += e.tcurr - e.wait_start;
        }
        e.state += 1;
        e.put_tnext(e.tcurr + e.get_delay(rng));
    } else if e.queue < e.max_queue {
        e.queue += 1;
    } else {
        e.failure += 1;
    }
}

pub fn out_act(e: &mut Element, rng: &mut SmallRng) {
    e.quantity += 1;
    e.state -= 1;
    if e.queue > 0 {
        e.queue -= 1;
        e.state += 1;
        e.put_tnext(e.tcurr + e.get_delay(rng));
    } else {
        e.wait_start = e.tcurr;
    }
    e.pop_tnext();
}

pub fn get_summary(e: &Element, nearest_tnext: String) -> String {
    format!(
        "\n
        ##### {} #####\n
        state = {} | quantity = {} | queue = {} | tnext = {} | average_load =
{:.4}\n
        failure = {}",
        e.name, e.state, e.quantity, e.queue, nearest_tnext, e.average_load,
e.failure
    )
}

pub fn do_statistics(e: &mut Element, delta: f64) {
    e.state_sum += (e.state as f64 * delta) as u32;
    e.mean_queue += e.queue as f64 * delta;
}

use std::cell::Cell;

use rocket::serde::{Deserialize, Serialize};

#[derive(Clone, Serialize, Deserialize, Debug)]
pub enum NextElementType {
```

```
    Balanced,
    RoundRobin(Cell<usize>),
    Random,
}

#[derive(Clone, Serialize, Deserialize, Debug, PartialEq)]
pub enum DistributionType {
    Exponential,
    Normal,
    Uniform,
    Erlang,
    Constant,
}

#[derive(PartialEq, Eq, Clone, Serialize, Deserialize, Debug)]
pub enum ElementType {
    Create,
    Process,
    Dispose,
}

use rand::Rng;
use rand::rngs::SmallRng;

pub fn exponential(rng: &mut SmallRng, mean: f64) -> f64 {
    let u: f64 = rng.random();
    -mean * u.ln()
}

pub fn normal(rng: &mut SmallRng, mean: f64, std_dev: f64) -> f64 {
    let u1: f64 = rng.random();
    let u2: f64 = rng.random();
    let z0 = (-2.0 * u1.ln()).sqrt() * (2.0 * std::f64::consts::PI * u2).cos();
    mean + std_dev * z0
}

pub fn uniform(rng: &mut SmallRng, a: f64, b: f64) -> f64 {
    let u: f64 = rng.random();
    a + (b - a) * u
}

pub fn erlang(rng: &mut SmallRng, mean: f64, k: usize) -> f64 {
    let mut product = 1.0;
    for _ in 0..k {
        let u: f64 = rng.random();
        product *= u;
    }
    -mean / k as f64 * product.ln()
}

pub fn round(x: f64, decimals: u32) -> f64 {
```

```
    let factor = 10f64.powi(decimals as i32);
    (x * factor).round() / factor
}
use cpu_time::ProcessTime;
use rand::seq::IndexedRandom;
use rand::SeedableRng;
use rand::rngs::SmallRng;
use rocket::serde::{Deserialize, Serialize};
use std::collections::VecDeque;
use std::fmt::Write;
use std::time::{Duration, Instant};
use sysinfo::{Pid, ProcessesToUpdate, System};

use crate::modeler::components::element::Element;
use crate::modeler::utils::consts::{ElementType, NextElementType};
use crate::modeler::utils::round::round;

#[derive(Debug)]
pub struct Model {
    elements: Vec<Element>,
    rng: SmallRng,
    iters: u32,
    tnext: f64,
    tcurr: f64,
    log_first: Vec<String>,
    log_last: VecDeque<String>,
    log_max_size: usize,

    mem_peak: f64,  // in KB
    mem_total: f64, // in KB
    mem_samples: u32,
    sample_interval: Duration,
    last_sample: Instant,
    sys: System,
    pid: Pid,
}

#[derive(Serialize, Deserialize, Debug)]
#[serde(crate = "rocket::serde")]
pub struct Results {
    results: Vec<SimSummary>,
    log_first: Vec<String>,
    log_last: VecDeque<String>,
    iters: u32,
    sim_time: f64,       // in seconds
    pub total_time: f64, // in seconds
    iter_per_sec: f64,
    mem_peak: f64, // in MB
    mem_mean: f64, // in MB
}
```

```
#[derive(Serialize, Deserialize, Debug)]
pub struct SimSummary {
    element: Element,
    quantity: u32,
    failures: u32,
    mean_queue_len: f64,
    fail_prob: f64,
    wait_time: f64,
}

impl Model {
    pub fn new(elements: Vec<Element>, log_max_size: usize) -> Self {
        Model {
            elements,
            rng: SmallRng::seed_from_u64(0),
            iters: 0,
            tnext: 0.0,
            tcurr: 0.0,
            log_first: Vec::with_capacity(log_max_size),
            log_last: VecDeque::with_capacity(log_max_size + 1), // +1 to account
for total sim results
            log_max_size,
            mem_peak: 0.0,
            mem_total: 0.0,
            mem_samples: 0,
            sample_interval: Duration::from_millis(500), // 0.5 seconds
            last_sample: Instant::now(),
            sys: System::new_all(),
            pid: Pid::from_u32(std::process::id()),
        }
    }

    pub fn simulate(&mut self, time: f64) -> Results {
        self.log_first.push(format!(
            "There are {} elements in the simulation",
            self.elements.len()
        ));

        let time_start = ProcessTime::now();
        self.mainloop(time);
        let time_elapsed = round(time_start.elapsed().as_secs_f64(), 4);

        self.log_sim_results();
        // trim extra newline
        self.log_last.get_mut(0).unwrap().remove(0);

        // print!("{:#?}", self.collect_sim_summary());
        Results {
            results: self.collect_sim_summary(),
            log_first: self.log_first.clone(),
            log_last: self.log_last.clone(),
```

```
            iters: self.iters,
            sim_time: time_elapsed,
            total_time: 0.0,
            iter_per_sec: round(self.iters as f64 / time_elapsed, 4),
            // in MB
            mem_peak: round(self.mem_peak / 1024.0, 4),
            mem_mean: round(self.mem_total / self.mem_samples as f64 / 1024.0,
4),
        }
    }

    fn mainloop(&mut self, time: f64) {
        // print!("{:#?}", self.elements);
        // preallocate memory for the vectors
        let mut to_out_act = Vec::with_capacity(self.elements.len());
        let mut to_in_act = Vec::with_capacity(self.elements.len());
        // thats it
        // thats the whole algorithm for ya
        while self.tcurr < time {
            // find next event
            self.tnext = f64::INFINITY;
            let mut event_id = 0;
            for e in &self.elements {
                let tnext_elem = e.get_tnext();
                if tnext_elem < self.tnext {
                    self.tnext = tnext_elem;
                    event_id = e.id;
                }
            }

            let tcurr_old = self.tcurr;
            self.tcurr = self.tnext;

            // update statistics
            for e in &mut self.elements {
                e.do_statistics(self.tcurr - tcurr_old);
                e.tcurr = self.tcurr;
            }

            // move things between relevant elements queues
            for (i, e) in self.elements.iter().enumerate() {
                if e.get_tnext() != self.tcurr {
                    continue;
                }
                // collect out_act elements
                to_out_act.push(i);
            }
            // collect in_act elements
            for i in &to_out_act {
                if let Some(in_id) = self.pick_in_act_element(i) {
                    to_in_act.push(in_id);
```

```
            }
        }
        // run actions
        for &i in &to_out_act {
            self.elements[i].out_act(&mut self.rng);
        }
        for &i in &to_in_act {
            self.elements[i].in_act(&mut self.rng);
        }
        to_out_act.clear();
        to_in_act.clear();

        // logging
        self.iters += 1;
        self.log_event(event_id);
        self.update_mem_stats();
    }
}

fn pick_in_act_element(&mut self, out_e_id: &usize) -> Option<usize> {
    let e = &self.elements[*out_e_id];

    if e.next_elements.len() == 1 {
        return Some(e.next_elements[0]);
    }

    match &e.next_element_type {
        NextElementType::Random => {
            if !e.next_elements.is_empty() {
                Some(*e.next_elements.choose(&mut self.rng).unwrap())
            } else {
                None
            }
        }
        NextElementType::RoundRobin(idx) => {
            if !e.next_elements.is_empty() {
                if idx.get() == e.next_elements.len() - 1 {
                    idx.set(0);
                } else {
                    idx.set(idx.get() + 1);
                }
                Some(*e.next_elements.get(idx.get()).unwrap())
            } else {
                None
            }
        }
        NextElementType::Balanced => {
            if !e.next_elements.is_empty() {
                let next_elements_copy = e.next_elements.clone();
                let mut min_queue_idx = 0;
                let mut min_queue = i32::MAX;
```

```
                    for i in next_elements_copy.iter() {
                        let next_elem = self.elements.iter().find(|e| e.id ==
*i).unwrap();
                        let free_queue = next_elem.queue as i32 - next_elem.state
as i32;
                        if free_queue < min_queue {
                            min_queue = free_queue;
                            min_queue_idx = *i;
                        }
                    }
                    Some(min_queue_idx)
                } else {
                    None
                }
            }
        }
    }

    fn log_event(&mut self, event_id: usize) {
        let e = &self.elements[event_id];
        let mut msg = String::with_capacity(512 + self.elements.len() * 128);

        // header
        write!(
            &mut msg,
            "\n
            >>>     Event #{} in {}     <<<\n
            >>>     time: {:.4}     <<<\n\n",
            self.iters, e.name, self.tnext
        )
        .unwrap();

        // element summaries
        for e in &mut self.elements {
            msg.push_str(&e.get_summary());
        }

        if self.log_first.len() <= self.log_max_size {
            self.log_first.push(msg);
        } else {
            if self.log_last.len() == self.log_max_size {
                self.log_last.pop_front();
            }
            self.log_last.push_back(msg);
        }
    }

    fn log_sim_results(&mut self) {
        // estimate capacity: small over-allocation to avoid reallocations
        let mut msg = String::with_capacity(1024 + self.elements.len() * 128);
```

```
        msg.push_str("\n\n-------------RESULTS-------------\n\n");

        for e in &self.elements {
            msg.push_str("##### ");
            msg.push_str(&e.name);
            msg.push_str(
                " #####\n
                quantity = ",
            );
            msg.push_str(&e.quantity.to_string());
            msg.push('\n');

            if e.elem_type == ElementType::Process {
                let failure_prob = if e.quantity + e.failure != 0 {
                    e.failure as f64 / (e.failure + e.quantity) as f64
                } else {
                    0.0
                };

                msg.push_str(&format!(
                    "\n
                    Mean length of queue = {:.4}\n
                    Failure probability = {:.4}\n
                    Total time stalling = {:.4}\n",
                    e.mean_queue / self.tcurr,
                    failure_prob,
                    e.wait_time
                ));
            }

            msg.push('\n'); // separate elements
        }

        msg.push_str(
            "--------------------------------\n
            Simulation is done successfully!",
        );

        self.log_last.push_back(msg);
    }

    fn collect_sim_summary(&self) -> Vec<SimSummary> {
        self.elements
            .iter()
            .map(|e| match e.elem_type {
                ElementType::Create | ElementType::Dispose => SimSummary {
                    element: e.clone(),
                    quantity: e.quantity,
                    failures: 0,
                    mean_queue_len: 0.0,
                    fail_prob: 0.0,
```

```
                    wait_time: 0.0,
                },
                ElementType::Process => SimSummary {
                    element: e.clone(),
                    quantity: e.quantity,
                    failures: e.failure,
                    mean_queue_len: e.mean_queue / self.tcurr,
                    fail_prob: if e.failure + e.quantity != 0 {
                        (e.failure as f64) / ((e.failure + e.quantity) as f64)
                    } else {
                        0.0
                    },
                    wait_time: e.wait_time,
                },
            })
            .collect()
    }

    fn update_mem_stats(&mut self) {
        if self.last_sample.elapsed() >= self.sample_interval {
            self.sys.refresh_processes(ProcessesToUpdate::All, true);
            if let Some(proc) = self.sys.process(self.pid) {
                let mem_curr = proc.memory() as f64 / 1024.0;
                self.mem_peak = self.mem_peak.max(mem_curr);
                self.mem_total += mem_curr;
                self.mem_samples += 1;
            }
            self.last_sample = Instant::now();
        }
    }
}

use rand::SeedableRng;
use rand::rngs::SmallRng;

use crate::modeler::utils::consts::DistributionType;
use crate::modeler::utils::random;

const SAMPLE_SIZE: u32 = 10000;

pub fn calculate_capacity(deviation: f64, dist: DistributionType, mean: f64, rep-
lica: u32) -> f64 {
    if dist == DistributionType::Constant {
        return replica as f64 / mean;
    }

    let mut rng = SmallRng::seed_from_u64(0);

    let iter = (0..SAMPLE_SIZE).map(|_| match dist {
        DistributionType::Exponential => random::exponential(&mut rng, mean),
        DistributionType::Normal => random::normal(&mut rng, mean, deviation),
```

```
        DistributionType::Uniform => random::uniform(&mut rng, mean - deviation,
mean + deviation),
        DistributionType::Erlang => random::erlang(&mut rng, mean, deviation as
usize),
        DistributionType::Constant => unreachable!(),
    });

    // average time per item (in seconds)
    let avg_time = streaming_mean(iter);

    replica as f64 / avg_time
}

fn streaming_mean<I>(iter: I) -> f64
where
    I: IntoIterator<Item = f64>,
{
    let mut sum = 0.0;
    let mut count = 0usize;
    for val in iter {
        sum += val;
        count += 1;
    }
    sum / count as f64
}

use std::cell::Cell;
use std::collections::HashMap;

use crate::modeler::components::element::{Element, reset_next_id};
use crate::modeler::utils::consts::DistributionType;
use crate::modeler::utils::consts::ElementType;
use crate::modeler::utils::consts::NextElementType;
use crate::routers::simulator::ElementInfo;

pub fn create_elements(model: HashMap<String, ElementInfo>) -> Vec<Element> {
    reset_next_id();
    let mut elements_by_id = HashMap::new();
    // initialize elements
    for (id, element_info) in &model {
        if !element_is_valid(&element_info) {
            continue;
        }
        let data = &element_info.data;
        let mut element = Element::new(
            data.replica,
            data.mean,
            data.deviation,
            parse_element_type(&element_info.class).unwrap(),
            parse_distribution(&data.dist).unwrap(),
            parse_next_element_type(&data.order).unwrap(),
```

```
            data.queuesize,
        );
        element.name = data.name.clone();
        elements_by_id.insert(id, element);
    }
    // chain elements together
    chain_elements(&model, &mut elements_by_id);
    let mut elements: Vec<Element> = elements_by_id.values().cloned().collect();
    // they need to be in order (ascending)
    elements.sort_by(|a, b| a.id.cmp(&b.id));
    elements
}

fn element_is_valid(element: &ElementInfo) -> bool {
    if element.inputs.is_empty() || element.outputs.is_empty() {
        true
    } else {
        element.inputs.len() > 0 && element.outputs.len() > 0
    }
}

pub fn parse_distribution(dist_name: &str) -> Option<DistributionType> {
    match dist_name.to_lowercase().as_str() {
        "exponential" => Some(DistributionType::Exponential),
        "normal" => Some(DistributionType::Normal),
        "uniform" => Some(DistributionType::Uniform),
        "erlang" => Some(DistributionType::Erlang),
        "constant" => Some(DistributionType::Constant),
        _ => None,
    }
}

pub fn parse_element_type(et_name: &str) -> Option<ElementType> {
    match et_name.to_lowercase().as_str() {
        "create" => Some(ElementType::Create),
        "process" => Some(ElementType::Process),
        "dispose" => Some(ElementType::Dispose),
        _ => None,
    }
}

pub fn parse_next_element_type(net_name: &str) -> Option<NextElementType> {
    match net_name.to_lowercase().as_str() {
        "balanced" => Some(NextElementType::Balanced),
        "round robin" => Some(NextElementType::RoundRobin(Cell::new(0))),
        "random" => Some(NextElementType::Random),
        _ => None,
    }
}

fn chain_elements(
```

```
    model: &HashMap<String, ElementInfo>,
    elements_by_id: &mut HashMap<&String, Element>,
) {
    // required to not crossrefrence the og elements_by_id
    let readonly_elements_by_id = elements_by_id.clone();
    // from output to input
    for (id, element_info) in model {
        if !element_is_valid(element_info) {
            continue;
        }

        let element = match elements_by_id.get_mut(id) {
            Some(el) => el,
            None => continue,
        };

        // dispose doesn't have an output
        if element.elem_type == ElementType::Dispose {
            continue;
        }

        element.next_elements = element_info
            .outputs
            .iter()
            .map(|out_id| readonly_elements_by_id.get(&out_id.to_string()).un-
wrap().id)
            .collect();
    }
}

use rocket::form::Form;
use rocket_dyn_templates::{Template, context};

use crate::modeler::model::Results;

#[get("/")]
pub fn index() -> Template {
    Template::render("index", context! {})
}

#[derive(FromForm)]
pub struct InputData {
    result: String,
}

#[post("/results", data = "<form_data>")]
pub fn results(form_data: Form<InputData>) -> Template {
    let data: Results = serde_json::from_str(&form_data.result).unwrap();
    Template::render("results", data)
}
```

```
use std::collections::HashMap;

use cpu_time::ProcessTime;
use rocket::serde::json::{Json, from_str};
use rocket::serde::Deserialize;

use super::utils::element_parser::{create_elements, parse_distribution};
use super::utils::capacity_calculator::calculate_capacity;
use crate::modeler::model::{Model, Results};
use crate::modeler::utils::round::round;

#[derive(Deserialize)]
#[serde(crate = "rocket::serde")]
pub struct SimRequest {
    model: HashMap<String, ElementInfo>,
    simtime: f64,
    log_max_size: u64,
}

#[derive(Deserialize)]
pub struct ElementInfo {
    pub class: String,
    pub data: ElementData,
    pub inputs: Vec<usize>,
    pub outputs: Vec<usize>,
}

#[derive(Deserialize)]
pub struct ElementData {
    pub deviation: f64,
    pub dist: String,
    pub mean: f64,
    pub name: String,
    pub order: String,
    pub queuesize: u32,
    pub replica: u32,
}

#[post("/simulate", format = "application/json", data = "<request>")]
pub fn simulate(request: Json<SimRequest>) -> Json<Results> {
    let time_start = ProcessTime::now();
    let data = request.into_inner();
    assert!(data.simtime > 0.0 && data.log_max_size > 0);

    let elements = create_elements(data.model);
    print!("Modeling started!");
    let mut simdata = Model::new(elements, data.log_max_size as usize).simu-
late(data.simtime);
    print!("Modeling finished!");
    let time_elapsed = round(time_start.elapsed().as_secs_f64(), 4);
```

```
    simdata.total_time = time_elapsed;
    // print!("{:#?}", simdata);
    Json(simdata)
}

#[derive(Deserialize)]
#[serde(crate = "rocket::serde")]
pub struct LoadRequest {
    deviation: f64,
    dist: String,
    mean: f64,
    replica: u32,
}

#[post("/capacity", data = "<request>")]
pub fn capacity(request: String) -> String {
    let data: LoadRequest = from_str(&request).unwrap();

    let capacity = calculate_capacity(
        data.deviation,
        parse_distribution(&data.dist).unwrap(),
        data.mean,
        data.replica,
    );

    capacity.to_string()
}
```