Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

УДК 004.942

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: "Засіб тестування IoT генераторів випадкових чисел з використанням багатовимірних статистик"

Спеціальність – 121 "Інженерія програмного забезпечення"

ПОЯСНЮВАЛЬНА ЗАПИСКА

БР.ІПЗ – 22.00.00.000

Студент

ІПЗ-44_____/Олексій РАЙЧЕВ/

Науковий керівник

к. ф.-м. н., доц. ____/Світлана ПОПЕРЕШНЯК/

Консультант

з питань нормоконтролю

фахівець_____/Тамара ЧАПОВСЬКА/

Допускається до захисту

Завідувач кафедри

д.т.н., проф._____/Олексій БИЧКОВ/

Київ – 2021

Рішенням Екзаменаційної комісії
випускна кваліфікаційна робота студента

_____

захищена з оцінкою

_____

Голова Екзаменаційної комісії
д. т. н., проф. Андрій БОНДАРЧУК

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖЕНО
Зав. кафедри програмних систем і технологій
_____(Олексій БИЧКОВ)
„_____”_____ 2021 р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ
СТУДЕНТУ
Райчеву Олексію Олеговичу

1. Тема випускної кваліфікаційної бакалаврської роботи “ Засіб тестування IoT генераторів випадкових чисел з використанням багатовимірних статистик”, керівник роботи Світлана ПОПЕРЕШНЯК, к. ф.-м. н., доц. затверджені на засіданні кафедри програмних систем і технологій, протокол №6 від «11» листопада 2020р.
2. Строк здачі студентом закінченої роботи _____
3. Вихідні дані до роботи: статті та тези конференцій вітчизняних і зарубіжних авторів, інтернет ресурси з питань тестування чисел на випадковість та IoT генераторів випадкових чисел, мови програмування, інтегровані середовища розробки, програмне забезпечення для створення діаграм та проектування.
4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)

1. Постановка задачі.
2. Тестування бітових послідовностей на основі багатовимірних статистик.
3. Інженерія програмного забезпечення.
4. Розробка моделі генератора випадкових чисел.
5. Тестування моделі генератора випадкових чисел.
6. Інструкція користувача.

5. Перелік графічного матеріалу (з точним зазначенням обов’язковихкреслень)

1. Спрощені діаграми класів для пакетів бібліотеки (рис. 3.1 та рис. 3.2)
2. Діаграма класів для API (рис. 3.3)

3. Діаграма діяльності алгоритму викликів API (рис. 3.4)

4. Діаграма варіантів використання для веб-додатку (рис. 3.5)

5. Діаграма діяльності для запитів зроблених через веб-додаток (рис. 3.6)

6. Макетна схема та фізична модель генератора (рис. 4.1)

7. Ілюстрація зміни даних на датчиках в часі (рис. 5.1)

8. Алгоритм програми ГПЧ на Arduino (рис. 5.2)

9. Конфігурація середовища і використання бібліотеки (рис. 6.1 та рис. 6.2)

10. Ілюстрація використання веб додатку (рис. 6.3-6.9)

6. Консультанти з роботи із зазначенням розділів роботи, що їх стосуються

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
РОЗДІЛ 2	Світлана ПОПЕРЕШНЯК		
РОЗДІЛ 5	Світлана ПОПЕРЕШНЯК		

7. Дата видачі завдання «02» листопада 2020 р.

Керівник _____ (Світлана ПОПЕРЕШНЯК)

Завдання прийняв до виконання _____(Олексій РАЙЧЕВ)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів бакалаврської роботи	Термін виконання етапів роботи	Примітка
1	Аналіз літератури	10.01.2021	виконано
2	Аналіз існуючих методів тестування послідовностей на випадковість	25.01.2021	виконано
3	Розробка пакету програм	01.03.2021	виконано
4	Тестування розробленого пакету програм	12.03.2021	виконано
5	Проектування і побудова моделі IoT генератора випадкових чисел	07.04.2021	виконано
6	Тестування моделі генератора та оптимізація	29.04.2021	виконано
7	Оформлення пояснювальної записки	20.05.2021	виконано
8	Оформлення презентації	30.05.2021	виконано

Студент – бакалавр_____ (Олексій РАЙЧЕВ)

Керівник роботи _____ (Світлана ПОПЕРЕШНЯК)

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 58 с., 19 рис., 13 табл., 5 додат., 9 джерел.

Тема: Засіб тестування IoT генераторів випадкових чисел з використанням багатовимірних статистик.

Об'єкт дослідження: IoT генератори випадкових чисел та випадкові послідовності бітів, створені за допомогою цих генераторів.

Мета роботи: Покращення існуючих та впровадження нових методів тестування бітової послідовності на випадковість. Дослідження практичного використання методів багатовимірних статистик до тестування якості IoT генераторів випадкових чисел.

Предмет дослідження: Методи тестування бітових послідовностей на випадковість з використанням багатовимірних статистик та методи NIST.

Результати дослідження:

Досліджено сучасні засоби для тестування бітових послідовностей на випадковість та виявлено проблеми в них. Запропоновано використання і формалізовано підхід до тестування з використанням багатовимірних статистик. Розібрано практичний випадок тестування та поліпшення генератора спираючись на результати отримані на стадії тестування і запропоновано підхід до перевірки якості генераторів.

Висновок:

Розроблено пакет програм для тестування на випадковість, що вирішує проблеми існуючих тестів за рахунок включення тестів заснованих на багатовимірних статистиках. Наданий практичний приклад використання пакету за рахунок реалізації фізичної моделі IoT генератора.

ВИПАДКОВІ ЧИСЛА, ТЕСТУВАННЯ, ГЕНЕРАТОР ВИПАДКОВИХ ЧИСЕЛ, БАГАТОВИМІРНІ СТАТИСТИКИ, INTERNET OF THINGS.

АННОТАЦИЯ

Выпускная квалификационная бакалаврская работа: 58 с., 19 рис., 13 табл., 5 прилож., 9 источников.

Тема: Средство тестирования IoT генераторов случайных чисел с использованием многомерных статистик.

Объект исследования: IoT генераторы случайных чисел и случайные последовательности битов, созданные с помощью этих генераторов.

Цель работы: Улучшение существующих и внедрение новых методов тестирования битовой последовательности на случайность. Исследование практического использования методов многомерных статистик к тестированию качества IoT генераторов случайных чисел.

Предмет исследования: Методы тестирования битовых последовательностей на случайность с использование многомерных статистик и методы NIST.

Результаты исследования:

Исследовано современные средства для тестирования битовых последовательностей на случайность и обнаружено проблемы в них. Предложено использование и формализирован подход к тестированию с использованием многомерных статистик. Разобрано практический случай тестирования и улучшения генератора, опираясь на результаты полученные на стадии тестирования и предложено подход к проверке качества генераторов.

Вывод:

Разработано пакет программ для тестирования на случайность, который решает проблемы существующих тестов при помощи включения тестов, основанных на многомерных статистиках. Представлен практический пример использования пакета за счет реализации физической модели IoT генератора.

СЛУЧАЙНЫЕ ЧИСЛА, ТЕСТИРОВАНИЕ, ГЕНЕРАТОР СЛУЧАЙНЫХ ЧИСЕЛ, МНОГОМЕРНЫЕ СТАТИСТИКИ, INTERNET OF THINGS.

SUMMARY

Graduation qualifying bachelor's thesis: 58 p., 19 illus., 13 tables, 5 appen., 9 sources.

Topic: A tool for testing IoT random number generators with the use of multidimensional statistics.

Object of research: IoT random number generators and random sequences generated by them.

Goal of work: Improvement and implementation of new testing methods of bit sequences for randomness. Research of practical use of multidimensional statistics methods for testing quality of IoT random number generators.

Subject of research: Methods of testing sequences for randomness with the use of multidimensional statistics and NIST methods.

Research results:

Modern tools for resting bit sequences for randomness have been studied and a number of problems have been identified with them. The use of multidimensional statistics has been proposed and formalized. The practical case of testing and improving a generator based on the data obtained during testing has been discussed and an approach to testing generator quality has been proposed.

Conclusion:

A software package for testing for randomness, that solves the problems of existing tests by including tests based on multidimensional statistics has been created. A practical case of software package usage was provided, by means of implementing a physical model of an IoT generator.

RANDOM NUMBERS, TESTING, RANDOM NUMBER GENERATOR, MULTIDIMENSIONAL STATISTICS, INTERNET OF THINGS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ГПЧ – Генератор псевдовипадкових чисел

АГПЧ – Апаратний генератор випадкових чисел

NIST – National Institute of Standards and Technology

STS – Statistical Test Suite

API – Application Programming Interface

HTTP – Hypertext Transfer Protocol

URL – Uniform Resource Locator

JSON – JavaScript Object Notation

IoT – Internet of Things

ПЗ – Програмне забезпечення

CSS – Cascading Style Sheets

ВСТУП

Використання випадкових чисел завжди мало своє місце в діяльності людини, але набуло широкого розповсюдження за рахунок розвитку інформаційних технологій. Випадкові числа лежать в основі таких напрямів як криптографія, програмування, симуляція, ігрові системи тощо. Завданням що покладають на випадкові числа найчастіше є забезпечення незалежності однієї процедури в системі від іншої. Таким чином, можна забезпечити непередбачуваність криптографічних ключів, механізми випадковості в комп'ютерних іграх та оптимізацію математичних та програмних методів, наприклад, метод Монте-Карло.

Випадковість є частиною реального світу, яку ми можемо спостерігати в будь-який момент часу, але для роботи реальної системи необхідно отримати випадкові числа штучним чином і в необхідних обсягах. Генератори, що використовують для цього створюють послідовності двійкових чисел і поділяються на два типи: генератори псевдовипадкових чисел (далі - ГПЧ) та апаратні генератори випадкових чисел (далі – АГВЧ).

ГПЧ – являють собою алгоритми що створюють послідовність чисел властивості якої наближаються до дійсно випадкової. Сукупність згенерована таким способом не є дійсно випадковою, адже вона повністю залежна від значення з якого починається робота алгоритму.

АГВЧ – це апаратні пристрої, що використовують певний фізичний процес для отримання чисел. Такі генератори засновані на мікроскопічних феноменах що створюють сигнали з випадковою періодичністю або випадковим чином, наприклад, тепловий шум або феномени квантової механіки. Непередбачуваність таких фізичних процесів, як правило, доведена теорією фізики.

Якість випадкової послідовності, або «дійсна випадковість» є її ключовою характеристикою, адже в переважній більшості областей застосування, саме від цього фактору залежить корисний ефект. Щоб виміряти випадковість використовуються набори відповідних статистичних тестів. Їх результати, вказують на характеристики, за допомогою яких можна зробити висновки про

випадковість: лінійна залежність між частинами послідовності, періодичні властивості, здатність до компресії, тощо.

Сфера Internet of Things стрімко розвивається завдяки глобальним трендам до інформатизації та автоматизації. Генератори випадкових чисел широко використовуються в IoT пристроях для генерації криптографічних ключів і таким чином допомагають захистити дані. Однак, обмеження цих пристроїв в розмірах і обчислювальних можливостях також накладають обмеження на методи генерації випадкових чисел.

Метою бакалаврської роботи є покращення існуючих та впровадження нових методів тестування бітової послідовності на випадковість. Це включає розробку формального опису статистичних тестів та реалізацію відповідних програмних продуктів. Також, робота ставить на меті дослідження практичного використання методів багатовимірних статистик до тестування якості послідовностей створюваних IoT генератором випадкових чисел.

Досягнення мети включає виконання наступних задач:

– Огляд математичного підґрунтя та принців тестування послідовностей на випадковість;

– Аналіз існуючих пакетів тестів;

– Формальний опис методів заснованих на багатовимірних статистиках і обґрунтування їх ефективності для перевірки коротких послідовностей;

– Створення пакету рекомендованих методів для тестування послідовностей на випадковість;

– Реалізацію комплексу програм для проведення тестувань бітових послідовностей;

– Огляд обмежень та підвалин, що впливають на загальний дизайн IoT генераторів і генераторів випадкових чисел в цілому;

– Створення фізичної моделі IoT генератора;

– Тестування та поліпшення моделі з використанням результатів тестів заснованих на багатовимірних статистиках.

Об’єктом дослідження є IoT генератори випадкових чисел та випадкові послідовності створені за допомогою цих генераторів.

Предметом дослідження є методи тестування бітових послідовностей на випадковість з використанням багатовимірних статистик та методи NIST.

Результати роботи було представлено на конференціях MSTIoE-7, що проходила 22-23 грудня 2020 та MSTIoE-8, що проходила 14 травня 2021 року. Робота також приймала участь в Всеукраїнській науково-технічній конференції “Застосування програмного забезпечення в інфокомунікаційних технологіях” 12 лютого 2021 року.

РОЗДІЛ 1
ПОСТАНОВКА ЗАДАЧІ

1.1. Існуючі методи тестування

Перевірка генератора або послідовності випадкових чисел на непередбачуваність є важливим процесом, щоб забезпечити правильну і ефективну роботу системи, яка буде їх використовувати. Для виконання цієї задачі існує широкий набір пакетів тестування, що надають теоретичну основу випробувань та їх програмну реалізацію. Незважаючи на великий вибір тестів, кожен з них перевіряє тільки одну певну характеристику, тому ніякий з можливих тестів або їх наборів не може дати стовідсоткової відповіді на питання про випадковість послідовності.

Зазвичай, тестування проводиться безпосередньо на створеній генератором послідовності, але деякі з існуючих тестів, наприклад з пакету Diehard, передбачають створення послідовності відповідно до заданої тестом специфікації. Також, в залежності від випробування, можуть використовуватися різні розподіли та властивості послідовності. Найчастіше, випробування засновано на використанні двох гіпотез:

– H_0 – гіпотеза, що послідовність яку тестують є дійсно випадковою;

– H_a – гіпотеза, що послідовність не є випадковою.

Під час тесту, обчислюється статистичне значення послідовності відповідно до специфікації. Приймаючи нульову гіпотезу як основну, рішення про випадковість приймається на основі порівняння отриманої статистики і теоретичних властивостей випадкової послідовності. Рівень впевненості в результатах тесту визначається статистичною значимістю (довірчім інтервалом), що може складати 0.1, 0.05 або 0.01 в залежності від сфери використання та початкових вимог до точності. Так, при значимості $\alpha = 0.1$ вірогідність коректності результату буде складати 99%. Після проведення тесту робиться один з наступних можливих висновків:

1. Прийняти гіпотезу H_0 коли послідовність дійсно є випадковою;

2. Прийняти гіпотезу H_0 коли послідовність не є випадковою (помилка першого типу);

3. Прийняти гіпотезу H_a коли послідовність є випадковою (помилка другого типу);

4. Прийняти гіпотезу H_a коли послідовність дійсно не є випадковою.

Далі буде проведено опис найвідоміших наборів тестів для перевірки ГПЧ, АГВЧ та бітових послідовностей. Варто зауважити, що деякі з методів випробувань в наборах збігаються, адже вони всі засновані на одному математичному підґрунті.

1.1.1. NIST Statistical Test Suite

NIST STS - специфікація та відповідна бібліотека на мові С, що були випущені Інститутом Стандартів та Технологій США. Пакет складається з 15 тестів для аналізу бітових послідовностей що були згенеровані ГПЧ або АГВЧ. Далі подано коротку інформацію про цілі та специфіку тестів, повний опис тестів доступний в [1].

– Частотний тест

В основі тесту лежить пошук пропорції між нулями та одиницями що входять в бітову послідовність. Ціль тесту – перевірити чи є відношення одиниць та нулів наближеним до того, яке було б у дійсно випадкової послідовності, тобто 0.5.

– Частотний тест у блоці

Тест заснований на визначенні пропорції між кількістю нулів та одиниць в блоках бітів фіксованої довжини які входять в досліджувану послідовність. Ціль тесту – перевірити наскільки відношення є наближеним до властивості дійсно випадкової послідовності.

– Тест на подібні послідовності

Тест полягає у знаходженні числа послідовних однакових біт та порівнянні отриманого значення з очікуваним для дійсно випадкової послідовності. Також, випробування вказує на характер осциляції між нулями та одиницями.

– Тест на найбільшу послідовність одиниць в блоці

В основі тесту лежить порівняння найдовших послідовностей одиниць в блоці зі значенням яке теоретично має дійсно випадкова послідовність. В цьому тесті закономірності між одиницями будуть вказувати і на закономірності між нулями, тому необхідний лише один тест.

– Тест рангів бінарних матриць

Тест полягає у перевірці рангів матриць створених з неперервних бітів послідовності. Мета випробування - перевірити наявність лінійної залежності між рядками фіксованої довжини що входять в послідовність. Цей тест також зустрічається в бібліотеці тестів Diehard.

– Спектральний тест

Тест полягає в пошуку кількості піків дискретної трансформації Фур'є вхідної послідовності. Ціль - визначити періодичні властивості послідовності, що можуть вказувати на відхилення від гіпотези про випадковість. Для прийняття рішення про випадковість кількість піків що перевищують границю в 95% повинна бути наближена до 5%.

– Тести на шаблони що перетинаються/не перетинаються

Тест заснований на знаходженні входжень шаблонів до блоків послідовності. Ціль тесту – виявити генератори чисел що створюють дуже багато однакових шаблонів.

– Універсальний тест Маурера

Ціль тесту – визначити чи може послідовність бути суттєво стиснута без втрати інформації за допомогою перевірки кількості бітів між подібними шаблонами. Послідовність яка добре піддається стисненню не є випадковою.

– Тест на лінійну складність

В основі тесту лежить пошук довжини регістру зсуву з лінійним зворотнім зв’язком. Випробування визначає чи є послідовність достатньо складною щоб бути дійсно випадковою. Чим менша довжина регістру, тим більша вірогідність що рядок бітів не є випадковим.

– Тест на послідовності та тест на приблизну ентропію

Тести полягають у пошуку частоти входження всіх шаблонів довжин M, M-1 та M+1 в бітовий рядок. Цілі тестів – визначити наскільки наближеними є частоти появи шаблонів до дійсно випадкової послідовності. Випадкові послідовності відповідають рівномірному розподілу, а тому частота появи кожного шаблону повинна бути однакова. Тест на послідовності використовує шаблони що не перетинаються, а тест на приблизну ентропію – ті що перетинаються.

– Тест кумулятивних сум

Мета тесту – визначити, чи є сукупна сума часткових послідовностей, що виникають в досліджуваній занадто великою або занадто малою щоб послідовність могла вважатися дійсно випадковою.

– Тест на довільні виключення

Тест полягає у знаходженні кількості циклів, що мають точно K відвідувань, у кумулятивній сумі випадкового блукання. Мета - визначити, чи відповідає кількість відвідувань певного стану протягом циклу тому, що можна було б очікувати від випадкової послідовності.

– Тест на довільні виключення з варіантом

В основі тесту лежить пошук загальної кількості відвідувань певного стану в кумулятивній сумі випадкового блукання. Мета цього тесту - виявити відхилення від очікуваного числа відвідування різних станів у випадковому блуканні.

1.1.2. Тести Diehard

Батарея статистичних тестів призначена для виміру якості ГПЧ та АГВЧ, що була створена Джорджом Марсаглія у 1995 році. В основі більшості тестів лежить використання генератора для побудови послідовності відповідно да наданої специфікації і порівняння її характеристик з очікуваними від випадкової. Деякі з наведених випробувань можна виділити в групи за подібністю, а інші являють собою один тест. Більше інформації про тести можна знайти в [2, 3].

– Тест днів народжень

В тесті генеруються m «днів народжень» в «році» що складається з n днів і визначаються інтервали між днями народжень. Для дійсно випадкової послідовності інтервали повинні мати експоненційний розподіл.

– Тест перестановок довжини 5, що перетинаються

Тест полягає в оцінці перестановок шаблонів з послідовності великої довжини. Ціль тесту – визначити наскільки відношення всіх варіантів перестановок відповідає теоретичному в дійсно випадковій послідовності (всі перестановки повинні з'являтися з однаковою вірогідністю).

– Тести мавп

Дані тести засновані на теоремі про нескінченну мавпу і пропонують розглядати послідовності з декількох біт як «слова». Підраховується кількість «слів» що перетинаються, а слова що не з'являються мають відповідати певному розподілу, щоб послідовність можна було вважати дійсно випадковою.

– Тести на підрахунок одиниць

В основі тестів лежить підрахунок кількості послідовних або обраних байтів. На основі отриманих даних, кожен з байтів перетворюється на літеру і відбувається порівняння частоти появи «слів» що складаються з 5-ти літер з даними дійсно випадкової послідовності.

– «Тест паркінгу»

Тест полягає у використанні генератора для додавання кіл розміром 1 од. до квадрату розміром 100 на 100 од.. Коло що було «успішно припарковано» не повинно перетинатися з іншими. Після 12000 спроб, кількість «успішних» кіл повинна відповідати нормальному розподілу.

– Тест на мінімальну відстань

Для тесту необхідно випадковим чином розташувати в квадраті 10000x10000 од. 8000 точок, і знайти мінімальну відстань між парами точок. Для дійсно випадкової величини відстані повинні мати експоненційний розподіл з певним середнім значенням.

– Тест випадкових сфер

В тесті генерується 4000 точок в кубі з ребром 1000 од., і в кожній точці будується сфера з радіусом що дорівнює мінімальній відстані до іншої точки. Об’єм найменшої сфери повинен мати експоненційний розподіл з певним середнім значенням щоб послідовність можна було вважати дійсно випадковою.

– Тест на стиснення

В основі тесту лежить розбиття $k=2^{31}$ на випадкові дробові числа. Припускають кількість ітерацій необхідну для того щоб k стало дорівнювати 1, та повторюють процедуру 100000 разів. Якщо отримані дані відповідають еталонному розподілу, то генератор вважається дійсно випадковим.

– Тест сум що перетинаються

Тест полягає у пошуку характеристик сум послідовності що складається з дробових чисел. Сукупності послідовних частин довжиною 100 додаються і для них визначається середнє значення та дисперсія. Якщо числові характеристики відповідають еталонним, то роблять висновок про те, що послідовність є дійсно випадковою.

– Тест на подібні послідовності

В тесті використовується послідовність дробових чисел для якої рахуються зростаючі та спадаючі послідовності бітів. Ціль тесту – перевірити наскільки кількості спадаючих і зростаючих бітів наближені до очікуваних значень для випадкової послідовності.

– Тест «Крепс»

Тест засновано на грі «Крепс» - проводиться симуляція 200000 партій, для кожної яких визначається кількість виграшів та кидків необхідних щоб закінчити гру. На основі відповідності еталонним даним, робиться висновок про випадковість послідовності.

1.1.3. TestU01

Об’ємна бібліотека тестів на мові C, що включає реалізацію ГПЧ, тести та батареї тестів. Всі випробування що надаються, поділені в групи відповідно до модулів програми [4]:

– smultin - тестування рівномірності та незалежності ГПЧ;

– sentrop - тести на основі дискретних та безперервних емпіричних ентропій;

snpair - тести на основі відстані між найближчими точками в вибірці n рівномірно розподілених точок в t розмірностях;

– sknuth - реалізує класичні статистичні тести для ГПЧ, описані в книзі Д.Кнута;

– smarsa - реалізує тести описані Д. Маргсалія, та його співробітниками;

– svaria – тести на рівномірність засновані на простій статистиці;

– swalk – тести засновані на випадковому блуканні через множину цілих чисел;

– scomp – тести засновані на еволюції лінійної складності по мірі зростання бітової послідовності і тест заснований на її стисненні;

– sspectral – тести засновані на спектральних методах;

– sstring – тести що застосовуються до рядків випадкових бітів за рахунок їх поєднання;

– sspacings – тести на основі функцій суми інтервалів між відсортованими спостереженнями вибірки рівномірних розподілів.

1.2. Проблеми методів

Розглянуті пакети статистичних тестів мають солідне математичне підґрунтя і готову програмну реалізацію. Будь-який з них можна використати для оцінки послідовності або генератора і мати високий рівень впевненості в якості результатів. Однак, в екосистемі статистичних тестів на випадковість можна виділити наступні тренди:

– Наявна велика кількість різних тестів та пакетів, що часто підходять до вирішення задачі з зовсім різних сторін;

– Відсутні чіткі лідери, тобто, тести які можна рекомендувати для вирішення більшості проблем;

– Неможливо отримати точний висновок про випадковість послідовності навіть після виконання всіх можливих тестів;

– Майже всі окремі пакети та тести мають деякі обмеження або недоліки.

Логічним буде висновок, що область перевірки випадковості далеко не є завершеною і потребує додаткового дослідження та покращення існуючих підходів. Відповідно до [5] проблемами більшості тестів є:

- Випробування потребують послідовності великої довжини;

Наприклад, мінімальна рекомендована довжина послідовностей для NIST варіюється від 100 до 10^6, а деякі з тестів Diehard потребують по 100-200 тисяч біт. Звісно, якість результату покращується при збільшенні вибірки в будь-якому статистичному дослідженні, але не існує альтернативи для перевірки коротких послідовностей.

- деякі з параметрів тестів неможливо змінити;

Це здебільшого стосується тестів Diehard, які потребують генерації послідовності фіксованої довжини відповідно до специфікації. Зміна параметрів має ключове значення для проведення якісного дослідження.

- рішення про проходження тесту приймає тільки два значення (так/ні);

Результатами тесту повинні також бути точні та значущі числові значення. Це дозволить порівнювати результати різних тестів або одного тесту для різних послідовностей.

- відсутність програмних пакетів для тестування.

Розробка пакетів для дослідження випадкових чисел без програмного забезпечення є доволі сумнівною роботою, адже область застосування повністю складається з інформаційних технологій.

Отже, в методах перевірки бітових послідовностей є достатньо проблем для вирішення та підходів для покращення. Особливий інтерес для дослідження складає відсутність тестів, що можуть дати адекватні результати на коротких послідовностях.

РОЗДІЛ 2
ТЕСТУВАННЯ БІТОВИХ ПОСЛІДОВНОСТЕЙ НА ОСНОВІ БАГАТОВИМІРНИХ СТАТИСТИК

2.1. Огляд методів тестування

Специфіка тестів описаних пакетів є такою, що на основі вхідної послідовності бітів визначається статистика яка або є результатом, або використовується для його пошуку. Цей підхід враховує тільки одну характеристику послідовності при одному випробуванні. Багатовимірні статистики орієнтовані на декілька властивостей, що дозволяє більш точніше оцінити коротку послідовність, але має свої недоліки в тестуванні довгої через надмірно велику кількість варіантів комбінацій статистик.

Відповідно до [1], генератори випадкових чисел мають тенденцію до створення великої кількості повторюваних шаблонів. Тести багатовимірних статистик, також, надають більш ефективні результати в перевірці шаблонів за рахунок оцінки декількох статистик одночасно.

Методи що представлені в роботі засновані на дослідженні кількості входжень двох- та трьох-бітових шаблонів в послідовність бітів. Тести на основі багатовимірних статистик в результаті виконання надають спільну вірогідність відповідної кількості шаблонів в послідовності заданої довжини. Той самий результат можна отримати за допомогою емпіричного підрахунку. Припустимо, що виконується розрахунок спільної вірогідності для всіх можливих значень $k_1 = \eta(00)$, $k_2 = \eta(111)$ та послідовності довжиною 3. Кількості входжень k_1 та k_2 до послідовності наведено в табл. 2.1.

Таблиця 2.1

Поява шаблонів в послідовності довжиною 3

Послідовність	**k_1**	**k_2**
000	2	0
001	1	0
010	0	0
011	0	0
100	1	0
101	0	0
110	0	0
111	0	1

Підрахувавши кількість появи для всіх можливих комбінацій k_1 та k_2, можна знайти відповідні вірогідності (табл. 2.2).

Таблиця 2.2

Входження шаблонів в послідовність довжиною 3

k_1	k_2	Кількість	Вірогідність
2	0	1	0,125
1	0	2	0,25
0	0	4	0,5
0	1	1	0,125

Емпіричним методом знайдено спільну вірогідність для заданої довжини і всіх можливих значень k. Цей підхід є доволі простим і наглядно показує для чого використовуються методи багатовимірних статистик, але не є ефективним (кількість послідовностей які необхідно перевірити при довжині 32 – 2^{32}). Випробування побудовані на формулах спільної вірогідності є більш доцільними як в математичному сенсі, так і в програмному.

Тести багатовимірних статистик відрізняються тільки шаблонами, на які перевіряється послідовність. Кожен метод отримує на вхід випадкову величину:

$$\gamma_1, \gamma_2 \ldots, \gamma_n \tag{2.1}$$

де $\gamma_i \in \{0, 1\}$, i = 1,2…,n, n > 0

Для даної величини визначається кількість специфічних шаблонів k_1, k_2 та k_3(якщо це визначено методом) і виконується обчислення за допомогою формули специфічної для методу.

Перший тест виконується, щоб знайти спільну вірогідність появи подій $k_1 = \eta(tt^*)$ та $k_2 = \eta(t1t^*) + \eta(t0t^*)$, при $t \in \{0, 1\}$, $t^*=1 - t$:

$$P\{\eta(tt^*) = k_1, \eta(t1t^*) + \eta(t0t^*) = k_2\} = \sum_{m_1=k_1}^{n-k_1} p^{m_1} q^{m_0} \sum \prod_{i=0}^{1} C_{k_1}^{\delta_i} C_{m_i-k_1}^{k_1-\delta_i} \tag{2.2}$$

де n – довжина бітової послідовності

p – вірогідність появи t

q – вірогідність появи t^* (q = 1 - p)

$m_0 = n - m_1$

$\sum$ - сума по всім комбінаціям δ_0 та δ_1, таким, що: $\delta_0 + \delta_1 = 2k_1 + k_2$

Другий метод тестування знаходить спільну вірогідність появи подій $k_1 = \eta(tt^*)$ та $k_2 = \eta(ttt^*)$:

$$P\{\eta(tt^*) = k_1, \eta(ttt^*) = k_2\} = \sum_{m_1=k_1}^{n-k_1} p^{m_1} q^{m_0} C_{k_1}^{k_2} C_{m_1-k_1}^{k_2} C_{m_0}^{k_1} \tag{2.3}$$

де n – довжина бітової послідовності

p – вірогідність появи t

q – вірогідність появи t^* (q = 1 - p)

$m_0 = n - m_1$

Третій метод оцінює вірогідність появи шаблонів $k_1 = \eta(tt^*)$, $k_2 = \eta(t1t^*)$ та $k_3 = \eta(t0t^*)$:

$$P\{\eta(tt^*) = k_1, \eta(t1t^*) = k_2, \eta(t0t^*) = k_3\}$$
$$= \sum_{m_1=k_1}^{n-k_1} p^{m_1} q^{m_0} C_{k_1}^{k_2} C_{k_1}^{k_3} C_{m_1-k_1}^{k_2} C_{m_0-k_1}^{k_3} \quad (2.4)$$

де n – довжина бітової послідовності

p – вірогідність появи t

q – вірогідність появи t^* (q = 1 - p)

$m_0 = n - m_1$

Четвертий метод використовується щоб оцінити спільну вірогідність подій $k_1 = \eta(tt^*)$ та $k_2 = \eta(t1t) + \eta(t0t)$:

$$\mathrm{P}\{\eta(tt^*) = k_1, \eta(t1t) + \eta(t0t) = k_2\}$$
$$= \sum_{m_1=k_1}^{n-k_1} p^{m_1} q^{m_0} \times \sum_{i \in \{k_1, k_1+1\}} \sum C_{i-1}^{\delta_0} C_i^{\delta_1 - m_1 + 2i} C_{m_0-i+1}^{k_1-\delta_0} \quad (2.5)$$
$$\times \mathrm{Z}(m_1 - i; m_1 - i - \delta_1)$$

де n – довжина бітової послідовності

$m_0 = n - m_1$

p – вірогідність появи t

q – вірогідність появи t^* (q = 1 - p)

$\sum$ - сума по всім комбінаціям δ_0 та δ_1, таким, що: $\delta_0 + \delta_1 = k_2$

Z(a, b) – визначається за допомогою формули 2.6

$$\mathrm{Z}(a, b) = \begin{cases} C_{a-1}^{b-1}, \text{якщо } a \geq b \geq 0; \\ 1, \text{якщо } a = b = 0; \\ 0, \text{в іншому випадку} \end{cases} \quad (2.6)$$

За допомогою п’ятого методу можна визначити вірогідність подій $k_1 = \eta(tt^*)$ та $k_2 = \eta(ttt)$:

$$\mathrm{P}\{\eta(tt^*) = k_1, \eta(ttt) = k_2\} = \sum_{m_1=k_1}^{n-k_1} p^{m_1} q^{m_0} C_{m_0}^{k_1} \times \sum_{i\in\{k_1,\, k_1+1\}} C_i^{m_1-k_2-i}\, \mathrm{Z}(m_1 - i, m_1 - i - k_2) \quad (2.7)$$

де n – довжина бітової послідовності

p – вірогідність появи t

q – вірогідність появи t* (q = 1 - p)

$m_0 = n - m_1$

Z(a, b) – визначається за допомогою формули 2.6

Шостий метод можна використати для того щоб знайти спільну вірогідність $k_1 = \eta(tt^*)$ та $k_2 = \eta(tt^*t)$:

$$P\{\eta(t\,t^*) = k_1, \eta(tt^*t) = k_2\} = \sum_{m_1=k_1}^{m-k_1} p^{m_1} q^{m_0} \sum_{i\in\{k_1,\, k_1-1\}} C_i^{k_2}\, C_{m_0-i}^{k_1-k_2} \times Z(m_1;\, i+1) \quad (2.8)$$

де n – довжина бітової послідовності

p – вірогідність появи t

q – вірогідність появи t* (q = 1 - p)

$m_0 = n - m_1$

Z(a, b) – визначається за допомогою формули 2.6

Сьомий метод шукає спільну вірогідність $k_1 = \eta(tt^*)$, $k_2 = \eta(ttt)$ та $k_3 = \eta(tt^*t)$:

$$\mathrm{P}\{\eta(t\,t^*) = k_1, \eta(ttt) = k_2, \eta(tt^*t) = k_3\} = \sum_{m_1=k_1}^{m-k_1} p^{m_1} q^{m_0} \times \sum_{i\in\{k_1,\, k_1+1\}} C_i^{k_2-m_1+2i}\, C_{i-1}^{k_3} C_{m_0-i+1}^{k_1-k_3} \times \mathrm{Z}(m_1 - i;\, m_1 - i - k_2) \quad (2.9)$$

де n – довжина бітової послідовності

p – вірогідність появи t

q – вірогідність появи t^* (q = 1 - p)

$m_0 = n - m_1$

Z(a, b) – визначається за допомогою формули 2.6

Восьмий метод виконується, щоб знайти спільну вірогідність подій $k_1 = \eta(tt^*)$, $k_2 = \eta(ttt)$ та $k_3 = \eta(tt^*t)$:

$$\begin{aligned} &P\{\eta(t\,t) = k_1, \eta(t^*tt^*) = k_2\} \\ &= \sum_{m_1=0}^{n} p^{m_1} q^{m_0} \left\{ C_{a-2}^{k_2} C_{k_1+1}^{a-k_2-1} Z(m_0;\ a-1) \right. \\ &+ C_a^{k_2} C_{m_0-1}^{a} Z(k_1;\ a-k_2) + 2C_{a-1}^{k_2} C_{k_1}^{a-k_2-1} C_{m_0-1}^{a-1} \\ &\left. + \chi(a-1 = k_2 = m_0 = 0) \right\} \end{aligned} \tag{2.10}$$

де n – довжина бітової послідовності

p – вірогідність появи t

q – вірогідність появи t* (q = 1 - p)

$m_0 = n - m_1$

χ(E) – індикатор події E

$a = m_1 - k_1$

Z(a, b) – визначається за допомогою формули 2.6

2.2. Порівняння тестів

Розглянемо результати виконання тестів NIST та багатовимірних статистик на коротких послідовностях за допомогою табл. 2.3. P_1 та P_2 були визначені за допомогою формул 2.2 та 2.7 відповідно. Спираючись на таблицю та [5] можна зробити висновок, що тести багатовимірних статистик дають кращі результати на коротких послідовностях.

Таблиця 2.3

Порівняння результатів

Тест Nist	Вхідні параметри	P-value	P_1	P_2
Frequency	1101010011	0.52708	0.14648	0.21191
Block frequency	1011100110 M = 2	0.84914	0.13671	0.18261
Runs	0011001111	0.59816	0.02050	0.18261
Longest run of ones	1101110111	0.00255	0.20507	0.18261
Binary matrix rank	11011101111101010011 M = 2	0.27012	0.09252	0.06847
Spectral	1011010101	0.46815	0.02343	0.04980
Non overlapping template	110101010111101100011 M = 10 T = 111	0.34415	0.05948	0.06863
Overlapping template	10010101101110010001 M = 10 T = 001	0.50979	0.06344	0.06055
Linear complexity	0011101101	0.99482	0.09765	0.21191
Maurers	000100111010011001O1 M = 2 L = 4	0.71	0.03172	0.06055
Entropy	0100100101 M = 3	0.40306	0.02734	0.04980

Продовження Таблиці 2.3

Serial	1101010011 M = 3	0.80879	0.14648	0.21191
Cumulative sums	1001001010	0.94173	0.14648	0.21191
Random excursions	1101010011	0.89307	0.11718	0.18554
Random excursion variant	0011100001	0.68309	0.13671	0.08691

РОЗДІЛ 3
ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Проектування і розробка

3.1.1. Бібліотека статистичних методів

Згідно з специфікацією NIST [1], та описаних методів багатовимірних статистик [5-7] створено бібліотеку що надає користувачам два інтерфейси для виконання відповідних статистичних тестів в мові програмування Java. Бібліотека складається з трьох пакетів зміст яких подано в табл. 3.1.

Таблиця 3.1

Пакети бібліотеки

Пакет	Зміст та опис пакету
nisttest	15 класів з тестами NIST, головний клас для виклику тестів
mdtest	8 класів з тестами заснованими на багатовимірних статистиках, головний клас для виклику тестів
util	Допоміжні класи, що включають: методи перевірки вхідних параметрів тестів; допоміжні методи для роботи з шаблонами і матрицями; функції erfc, Gamma та нормального розподілу; алгоритми дискретної трансформації Фур'є і Берлекампа-Мессі, та інші допоміжні функції що використовуються в обох пакетах тестів.

Інтерфейси для виклику методів є реалізаціями шаблону проектування «фасад» – вони групують окремі методи в один клас для уніфікації виклику методів та легкості використання. Спрощену діаграму класів для пакету NIST зображено на рис. 3.1, а повну в Додатку В. Головним інтерфейсом і «фасадом» є клас «NistTest», що одночасно виконує перенаправлення викликів до відповідних класів і перевірку вхідних параметрів за допомогою класу «Validator». Результатом, і значенням що повертає тест є об'єкт класу «TestResult» або список таких об'єктів коли

проводиться декілька тестів в одному методі. Кожен з окремих класів статистичних методів (наприклад, «Frequency»), використовує хоча б один клас з пакету «util» в своїй реалізації.

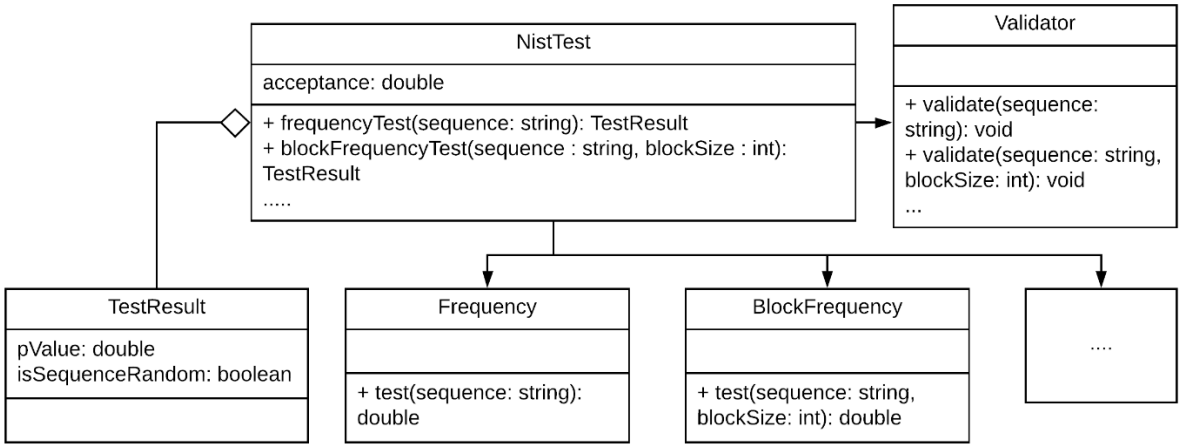

Рис. 3.1. Спрощена діаграма класів для тестів NIST

Окрім «фасаду», класи тестів багатовимірних статистик використовують шаблон «Шаблонний метод» для перевірки вхідних параметрів (рис. 3.2). Клас «MdStatistic» містить один абстрактний метод та один звичайний, а підкласи виконують визначення абстракції. При виклику однієї функції виконання передається іншій реалізованій нащадком. Кожен з тестів використовує засоби роботи з шаблонами класу «TemplateUtils» та біноміальний розподіл або функцію Z (формула 2.6) з класу «FunctionUtils». Повна діаграма класів знаходиться в Додатку В.

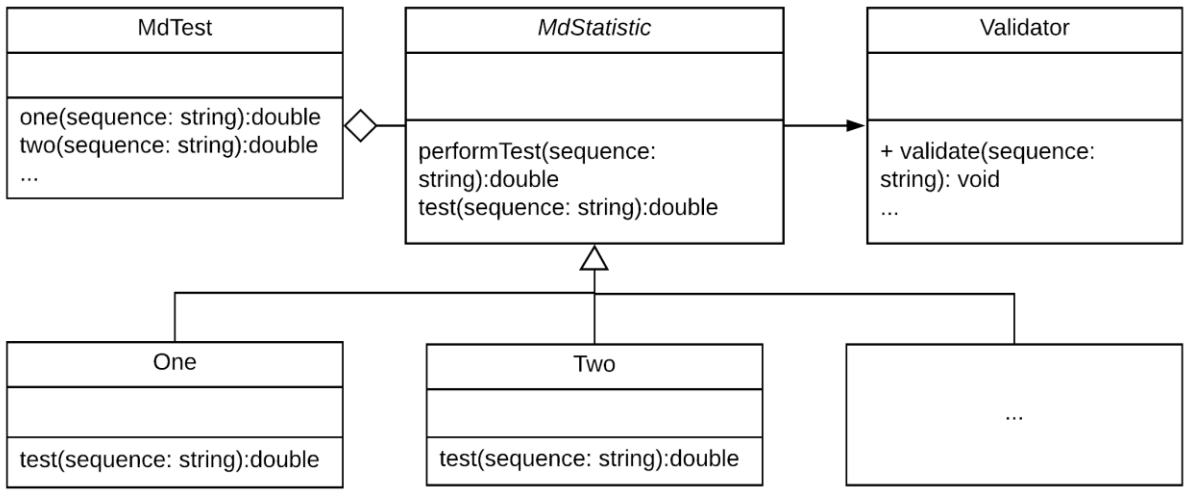

Рис. 3.2. Спрощена діаграма класів для тестів багатовимірних статистик

3.1.2. Сервер і прикладний програмний інтерфейс

Сервер програми виконує обробку вхідних HTTP запитів (диспетчеризацію, зчитування даних з формату JSON і передачу відповідним статистичним тестам) і створення відповідей (створення повідомлень про помилку, перетворення об’єктів у JSON, відправка відповіді). Пакетну структуру серверу подано в табл. 3.2.

Таблиця 3.2

Пакети серверу

Пакет	Зміст та опис пакету
controller	Класи «NistController» та «MdController», що визначають URL та HTTP методи доступні на сервері і викликають відповідні методи їх обробки
service	Інтерфейси і класи що забезпечують виконання бізнес-логіки серверу
util	Допоміжні класи, що включають: конвертери перелічуваних типів даних, обробник помилкових запитів, класи для виконання методів з бібліотеки статистичних тестів

Для всіх тестів з бібліотеки було створено чотири адреси виконання статистичних тестів (табл. 3.3). При надходженні запиту, виконується перевірка наявності обробника на сервері, і якщо його знайдено, запит передається відповідному контролеру який в свою чергу викликає сервіс що визначено в методі.

Таблиця 3.3

HTTP методи і адреси доступні на сервері

Метод	Відносна адреса	Опис функціональності
POST	/nist_test/{testName}	Параметр {testName} визначає статистичний тест який буде виконано, наприклад «/nist_test/binaryMatrix». За даними адресами виконується тільки один тест, який потребує правильних параметрів в тілі запиту
	/md_test/{testName}	

Продовження Таблиці 3.3

Метод	Відносна адреса	Опис функціональності
POST	/nist_test	Виконуються всі тести NIST JSON-об’єкти з параметрами яких наявні в тілі запиту.
	/md_test?tests=[testName1, testName2…]	Виконуються всі тести багатовимірних статистик імена яких наявні в параметрах запиту для послідовності з тіла запиту.

Імена тестів, які необхідно виконати визначені параметрами URL або тілом запиту. Щоб обрати відповідний метод, в програмі створено два класи – «NistFactory» та «MdFactory». Вони реалізують шаблони проектування «Фабрика» і «Шаблонний метод» за допомогою перелічуваного типу. Архітектуру основних компонентів модуля показано на рис. 3.3.

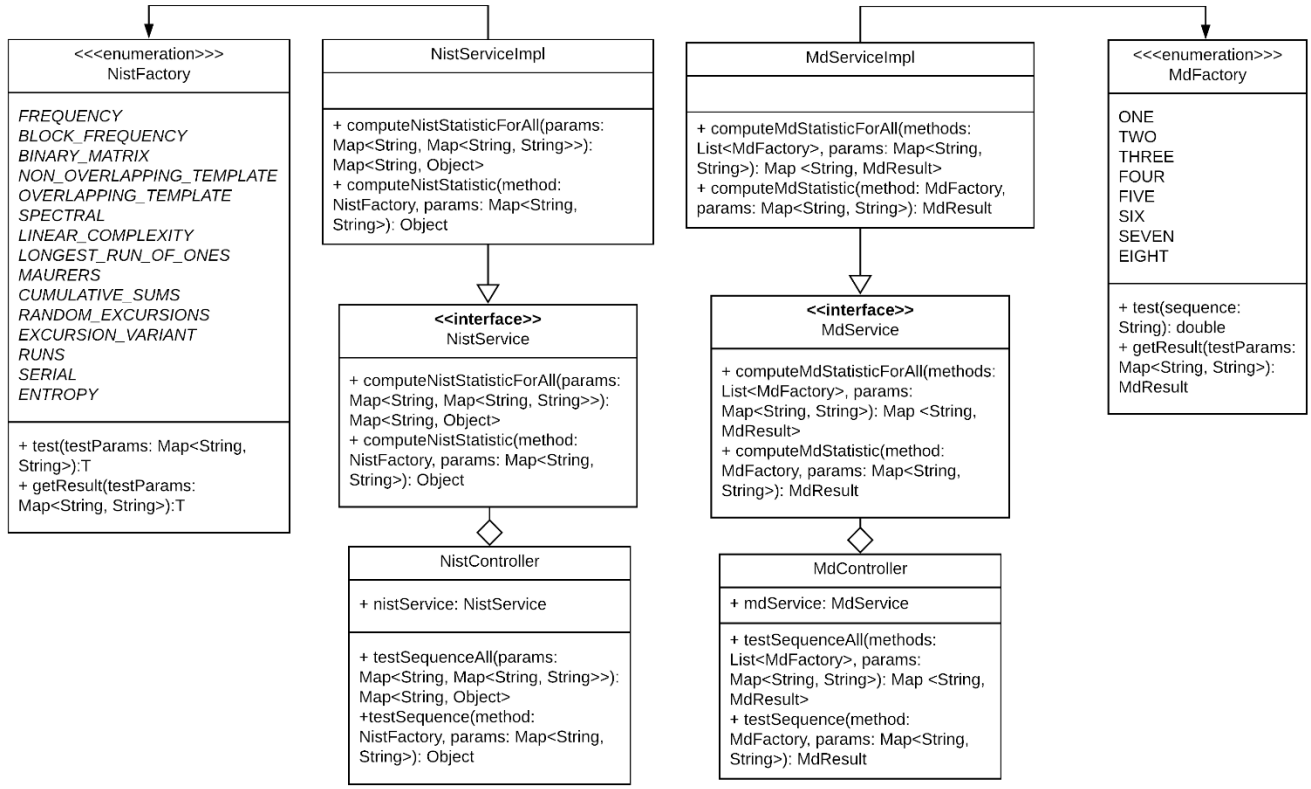

Рис. 3.3. Діаграма класів для основних модулів сервера

Враховуючи, що обробка вхідних запитів є найважливішою задачею серверу, розглянемо типовий сценарій за допомогою діаграми діяльності (рис. 3.4). Як

можна бачити, процес є доволі складним, і містить багато етапів на яких можуть виникнути критичні та помилкові ситуації.

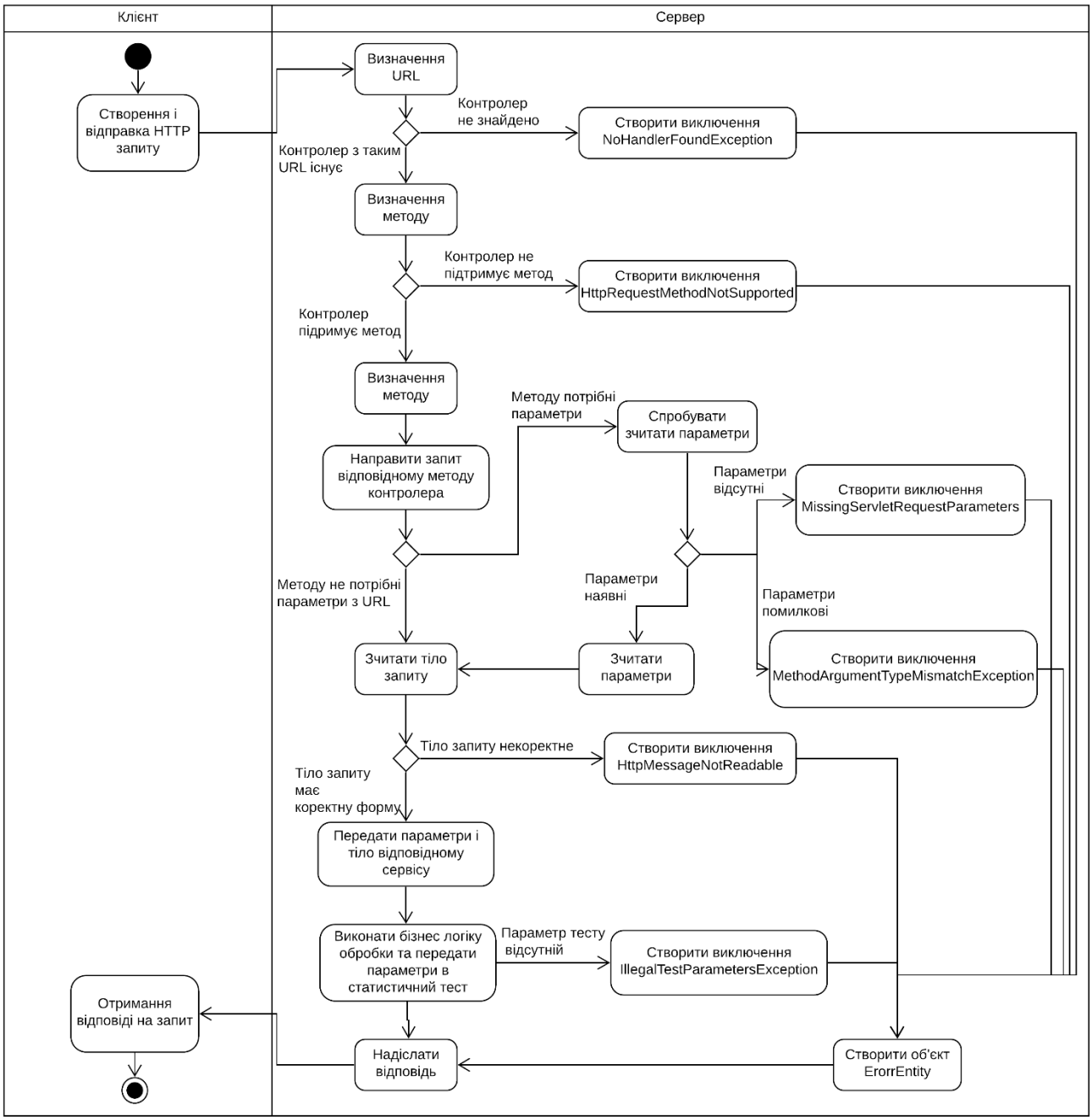

Рис. 3.4. Діаграма діяльності

Щоб забезпечити відповідний рівень параметрів надійності та робастності серверу, створено клас «ErrorHandler», який виконує перехоплення визначених виключень під час виконання запиту. Клас «ErrorEntity», що є контейнером для параметрів помилки, стандартизує вигляд і зміст повідомлень. При виникненні виключних ситуацій (табл. 3.4) повертається HTTP код, що відповідає їх змісту,

його розшифровка, текстове повідомлення, поточний час та URL. У випадку успішного виконання запиту, повертається тіло з результатами тесту та код 200 (OK).

Таблиця 3.4

HTTP коди і помилкові ситуації

Код	Текстовий опис	Помилкова ситуація і зміст повідомлення
404	Not Found	Виконання запиту з будь-яким методом за URL що не підтримується контролерами програми
400	Bad Request	Відсутність параметрів в URL
		Пусте тіло запиту
		Неправильна змінна шляху в URL
405	Method Not Allowed	Виконання запиту з методом що не підтримується за URL що підтримується програмою
422	Unprocessable Entity	Вхідні параметри статистичного тесту не є коректними

Створений сервер забезпечує прийом запитів, відправку відповідей, передачу вхідних параметрів відповідним статистичним тестам і обробку всіх можливих помилкових ситуацій.

3.1.3. Веб-додаток

Ціллю веб-додатку є забезпечити користувачів можливістю тестування бітових послідовностей на випадковість за допомогою графічного інтерфейсу. Він повинен забезпечити високий рівень зручності використання: надавати свободу дій, врахувати можливість помилок, повідомляти інформацію про стан системи та містити довідкові матеріали.

Основний функціонал який повинен забезпечувати додаток подано на діаграмі варіантів використання (рис. 3.5). Відповідно до діаграми, між задачами, що виконує веб-додаток та задачами серверу є чіткий розподіл. Перший працює незалежно більшу частину часу, і викликає другого тільки коли не може виконати поставлену задачу самостійно.

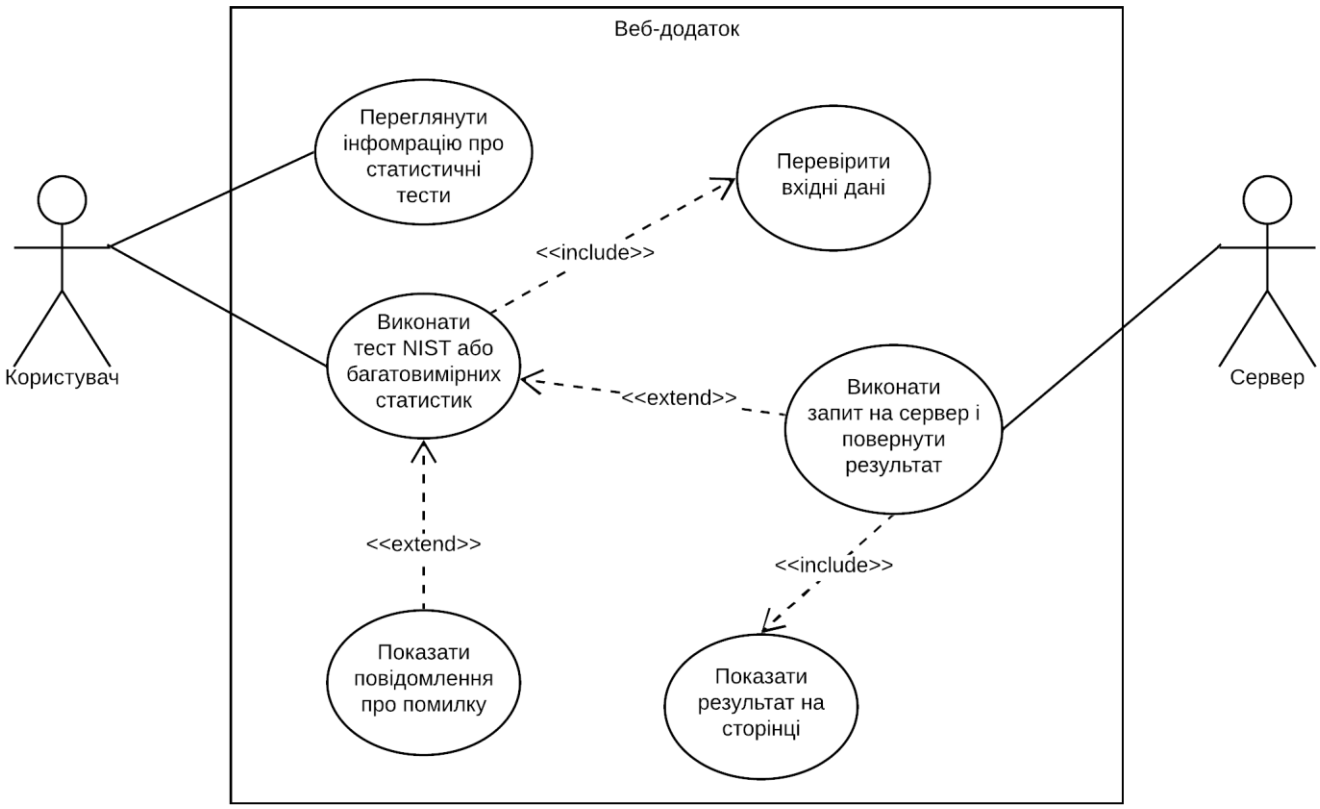

Рис. 3.5. Діаграма варіантів використання

Веб-сайт складається з однієї «сторінки», відповідно до принципів створення front end за допомогою React. Так, верхній та нижній та нижній колонтитули залишаються незмінними незалежно від URL, а зміст який знаходиться посередині завжди змінюється в залежності поточної адреси. Верхня навігаційна панель надає можливість перейти на 3 сторінки, зміст яких описано в табл. 3.5.

Таблиця 3.5

Зміст сторінок навігаційної панелі

Назва сторінки	Опис сторінки
Головна сторінка	Загальна інформація про тести та бітові послідовності і про призначення програмного продукту
Сторінка тестів NIST	Містить 16 сторінок що відповідають окремим тестам NIST та сторінку з комплексним тестом.
Сторінка тестів багатовимірних статистик	Містить 9 сторінок що відповідають окремим тестам багатовимірних статистик та сторінку з комплексним тестом.

Всі сторінки з тестами містять додаткове меню в якому можна обрати один окремий, або комплексний тест. На сторінці останнього, користувачу надається можливість ввести вхідні параметри, та вибрати один або декілька методів одночасно. Сторінки окремих тестів містять форми введення даних і теоретичну інформацію.

Компоненти веб-додатку можна умовно поділити на декілька груп:

– Компоненти тестів NIST – визначають елементи форми введення даних, адресу для прикладного інтерфейсу, довідковий матеріал до тестів і засоби перевірки вхідних параметрів;

– Компоненти багатовимірних статистик – визначають адресу для отримання даних з серверу та довідковий матеріал;

– Компоненти форм – групують спільні характеристики методів, для того, щоб прибрати повторюваність коду. Забезпечують перевірку і форматування параметрів за визначеними методами, реалізують створення, відправку та обробку відповіді HTTP запиту;

– Компоненти меню – визначають адреси за якими знаходяться всі сторінки, забезпечують відображення посилань на сторінках і правильний перехід по ним;

– Компоненти графіків – забезпечують побудову графіків відповідно до результатів тестування послідовності;

– Допоміжні компоненти – підтримують уніфіковану структуру результатів тестування та логіку переходу за посиланнями;

– Валідатор – об’єднує логіку перевірки вхідних параметрів, щоб знизити повторюваність коду;

– Стилі CSS – визначають зовнішній вигляд сторінок.

Загальний принцип спільної роботи пакету програм представлений на діаграмі діяльності (рис. 3.6). Як можна бачити, виконання навіть одного тесту не є тривіальною задачею, що включає багато етапів з використанням всіх модулів. За рахунок чітко визначених протоколів та інтерфейсів комунікації між модулями системи, та «лінивого виконання» всіх етапів, досягається високий рівень ефективності та надійності системи.

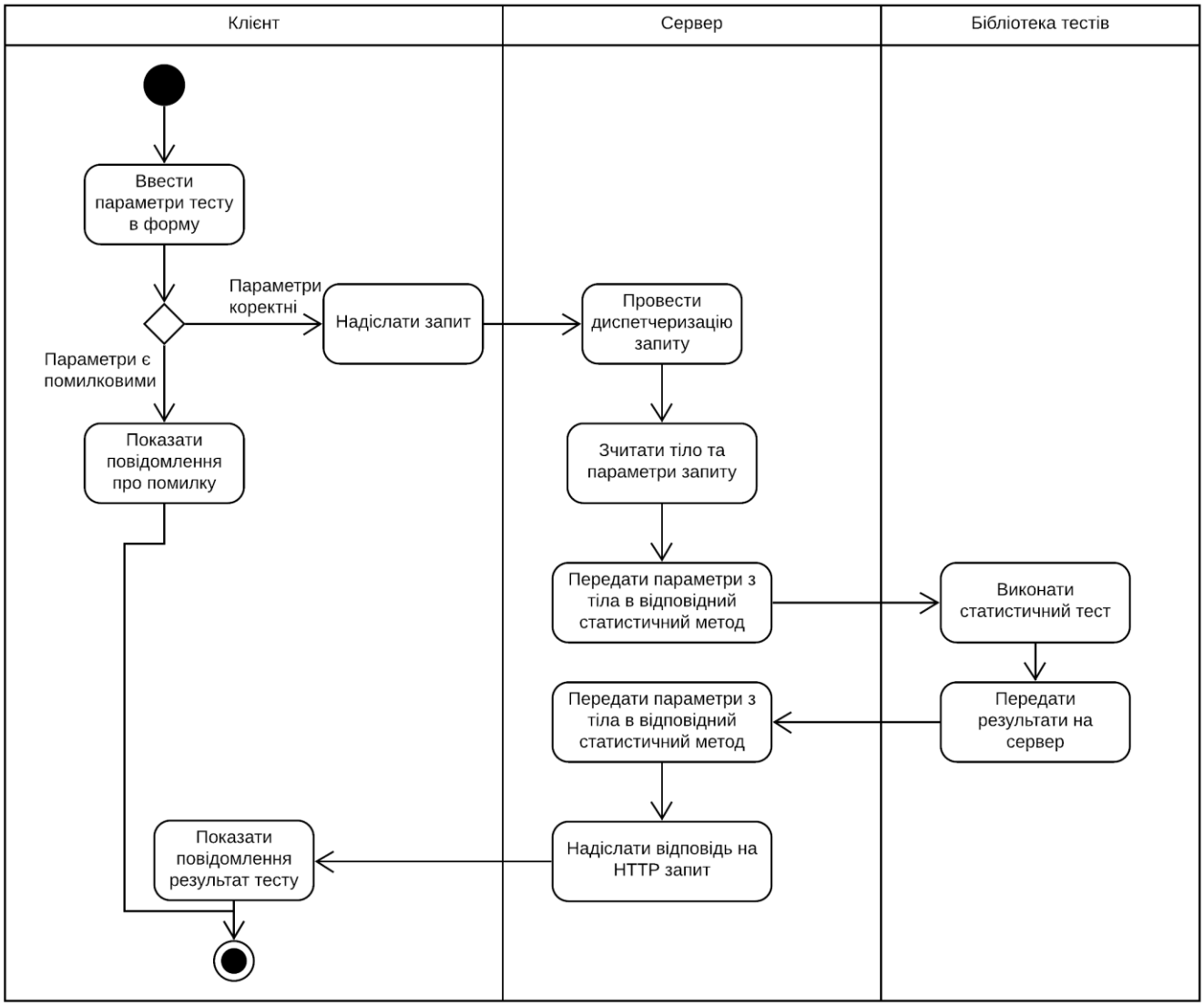

Рис. 3.6. Діаграма діяльності для повного циклу запит-відповідь

3.2. Тестування

Пакет програм складається з трьох модулів, що можуть бути використані на різних рівнях інтеграції. Так, є можливість додати бібліотеку в будь-який проект на мові java незалежно від інших. Якщо необхідно проводити обчислення на віддаленому сервері, або є необхідність створити веб-додаток – стане в нагоді прикладний інтерфейс. Для звичайних користувачів достатньо зайти на сайт і ввести параметри тесту в форму. Отже, щоб забезпечити якість пакету програм на всіх рівнях використання, необхідно провести комплексне тестування, що буде включати модульне, інтеграційне та системне.

3.2.1. Модульне тестування

При тестуванні методів з пакету «util» бібліотеки, використано принцип «висхідного тестування» - спочатку перевірялися термінальні функції, а на їх основі всі інші. Щоб забезпечити правильність результатів використано таблиці функцій (erfc, нормальний розподіл) та спеціальні математичні пакети (трансформація Фур'є, алгоритм Берлекампа-Мессі). Вибір вхідних параметрів проходив за допомогою побудови керуючих графів, визначення критичних значень та еквівалентного розбиття. Функції класу «Validator» перевірено на всіх можливих некоректних аргументах.

Тестування методів NIST та багатовимірних статистик проводилося з використанням «випадкового тестування», тобто, методи виконувалися на псевдовипадкових вхідних параметрах, а результат про успішність тесту приймався на основі відповідності результату в області допустимих значень. Для пакету NIST, результат повинен знаходитися в межах 0 до 1, а багатовимірні статистики дають відповідь ідентичну емпіричним підходам. За рахунок тестування на випадкових даних, набуто впевненості у відсутності критичних помилок, як: вихід за межі масиву, ділення на нуль, передача неправильних параметрів в інший метод, тощо. Також, для тестів NIST виконано перевірку на даних з прикладів обчислень представлених в [1].

Перевірка модулів серверу проходила за рахунок симуляції його роботи і виконання вхідних запитів. Через те, що всі кінцеві точки прикладного інтерфейсу не працюють без бібліотеки, модульне тестування виконано тільки на помилкових запитах, типи яких представлено в Таблиці 3.4.

Перевірка сайту включала забезпечення коректної роботи всіх посилань та елементів форми і випробовування форм на неправильних вхідних параметрах.

Всього створено 159 тест-кейсів, розроблено 120 автоматичних тестів і виконано 39 ручних.

3.2.2. Інтеграційне тестування

Інтеграційне тестування програм включало підключення бібліотеки до серверу і виконання HTTP запитів на сервер-симуляцію. Перевірка виконувалась безпосередньо для URL та методів що визначаються контролерами прикладного інтерфейсу та функціями API. Створено 7 тест-кейсів та автоматичних тестів для перевірки виконання статистичних тестів через HTTP запити.

3.2.3. Системне тестування

Системне тестування проводилося ручним способом, тобто, вхідні дані тестів вводилися в форми і виконувалася перевірка коректності запитів і результатів. Підбір вхідних даних проходив з прикладів обчислень, що представлені в [1] та тест-кейсів для модулів бібліотеки. В результаті проведено 25 тестів – по одному для комплексних методів і кожного статистичного методу окремо.

РОЗДІЛ 4
РОЗРОБКА МОДЕЛІ ГЕНЕРАТОРА ВИПАДКОВИХ ЧИСЕЛ

4.1. Підвалини до розробки генератора випадкових чисел

Програмний продукт, що було створено в цій роботі може використовуватися для вирішення широкого спектру задач, як уже і було неодноразово зазначено. Одною з найважливіших, та дійсно тою, що може отримати неоціненну користь сферою застосування є криптографія.

Завдяки сучасному розвитку технологій і високому рівню їх інтеграції у різні сфери людської діяльності з’являється необхідність створення програмного забезпечення для різних видів пристроїв з різними операційними системами та обчислювальними можливостями. Звісно, характеристики пристрою чи системного ПЗ не повинні впливати ні на якість, ні на продуктивність, ні на безпеку що надається самим пристроєм та ПЗ. Одним з «вузьких місць» сучасних IoT пристроїв є криптографія, адже ці пристрої здебільшого працюють в умовах обмежених обчислювальних можливостей, на що впливають такі фактори як розмір, можливість автономної роботи та навіть розмір батареї. Більшість алгоритмів шифрування базуються на доволі складних розрахунках і генерації випадкових чисел, що не може не впливати на необхідні характеристики пристрою. Використання програмного продукту для створення та тестування легковагових генераторів випадкових чисел може внести неоціненний вклад в розвиток не тільки IoT, а і всієї криптографічної сфери.

Створення будь-якого пристрою або програми починається з дизайну і визначення ключових факторів та обмежень, які дозволять продукту розвиватися у правильному напрямку та надати очікувані результати. Для генераторів випадкових чисел, першим важливим фактором є постановка задачі яку повинен вирішувати генератор.

Сказане в цьому розділі доволі повно описує актуальність проблеми легковагових генераторів в IoT пристроях, тому саме для вирішення цієї проблеми буде створено генератор.

Наступним важливим питанням є вибір між типами генераторів. ГПЧ має свої переваги для поставленої задачі, адже для його роботи необхідна тільки схема, що буде виконувати алгебраїчну функцію та деяка пам’ять за допомогою якої можна зберегти початкове або проміжне значення алгоритму. Недоліком цього типу генераторів є те, що він створює випадкові числа набагато гіршої якості (з більшою детермінованістю).

АГВЧ потребує значно більшої кількості технічного забезпечення – йому необхідна більша кількість пам’яті (для збереження проміжних даних), більш вимогливий процесор, різноманітні датчики (для вимірювання фізичних процесів), а іноді навіть підсилювач. Перевагою цього типу є те, що створювані числа мають значно нижчу ступінь детермінованості.

Вибір між генераторами є доволі адекватним питанням, не зважаючи на погані характеристики випадковості ГПЧ, адже в залежності від умов навіть ці генератори можуть надавати достатній рівень безпеки. Однак, якщо вирішення проблеми направлене на високі стандарти криптографічної безпеки і відповідно низьку детермінованість, вибір падає на АГВЧ.

Наступним питанням дизайну що стосується саме АГВЧ є вибір сенсорів що будуть використовуватися для отримання випадкових чисел з навколишнього середовища. В залежності від існуючих варіантів використання генератора, необхідно чітко визначитися з датчиками, адже датчик світла не буде корисним, якщо пристрій використовується в темному приміщенні, а датчик руху або акселерометр буде надлишковим для стаціонарного пристрою. Це є дуже суттєвим питанням при розробці реальних девайсів, але для створюваної моделі генератора це не є вирішальним фактором.

Іншими не менш важливими при для IoT пристроїв характеристиками є вага, розміри, інтеграція з програмним забезпеченням та іншими пристроями і т.д. Не зважаючи на серйозність цих факторів при розробці пристроїв для кінцевого

користувача, робота базується на тому щоб показати більш ефективний підхід до створення легковагових генераторів з використанням створеного програмного продукту для тестування та налагодження, тому ними можна частково чи повністю знехтувати.

4.2. Дизайн генератора випадкових чисел

На дизайн генератора більше за все вплинула постановка задачі. Так, типом генератора для моделі було обрано АГВЧ, завдяки тому, що його характеристики дозволять генерувати більш випадкові послідовності. Це рішення в свою чергу викликає необхідність створення фізичної моделі що буде складатися з мікросхеми та декількох сенсорів.

За основу генератора та мікросхему було вирішено використати плату Arduino. Факторами що вплинули на цей вибір є:

- Можливість використовувати високорівневу мову програмування;
- Наявність великої кількості сенсорів та додаткових пристроїв що можуть бути використані для створення моделі;
- Широка підтримка та велика кількість навчальних матеріалів;
- Можливість легко вносити зміни та модифікації в модель;
- Подібність Arduino до контролеру (в порівнянні з такими пристроями як Raspberry Pi), а отже більша подібність моделі до реальної схеми.

Використання Arduino надає широкий вибір сенсорів що можуть бути використані для отримання випадкових чисел з навколишнього середовища. Для моделі було обрано датчик звуку та акселерометр. Основним факторами вибору є:

- Легкість збору даних в звичному середовищі;
- Висока придатність даних сенсорів до використання в генераторах випадкових чисел [8];
- Широке використання даних сенсорів у сучасних пристроях.

Макетну схему та реальну модель генератора можна побачити на рисунку 4.1. Дана схема не відображає повну методику підключення елементів між собою, але

дає високорівневий огляд компонентів та взаємодії між ними. Різними кольорами на схемі позначено загальні види підключень елементів. Фіолетовий колір – напруга, жовтий – заземлення, а помаранчевий – передача даних. На реальній моделі можна побачити сенсори під'єднані до макетної дошки, саму плату Arudino та кнопку. Положення елементів на моделі відповідають їх положенням на макетній схемі.

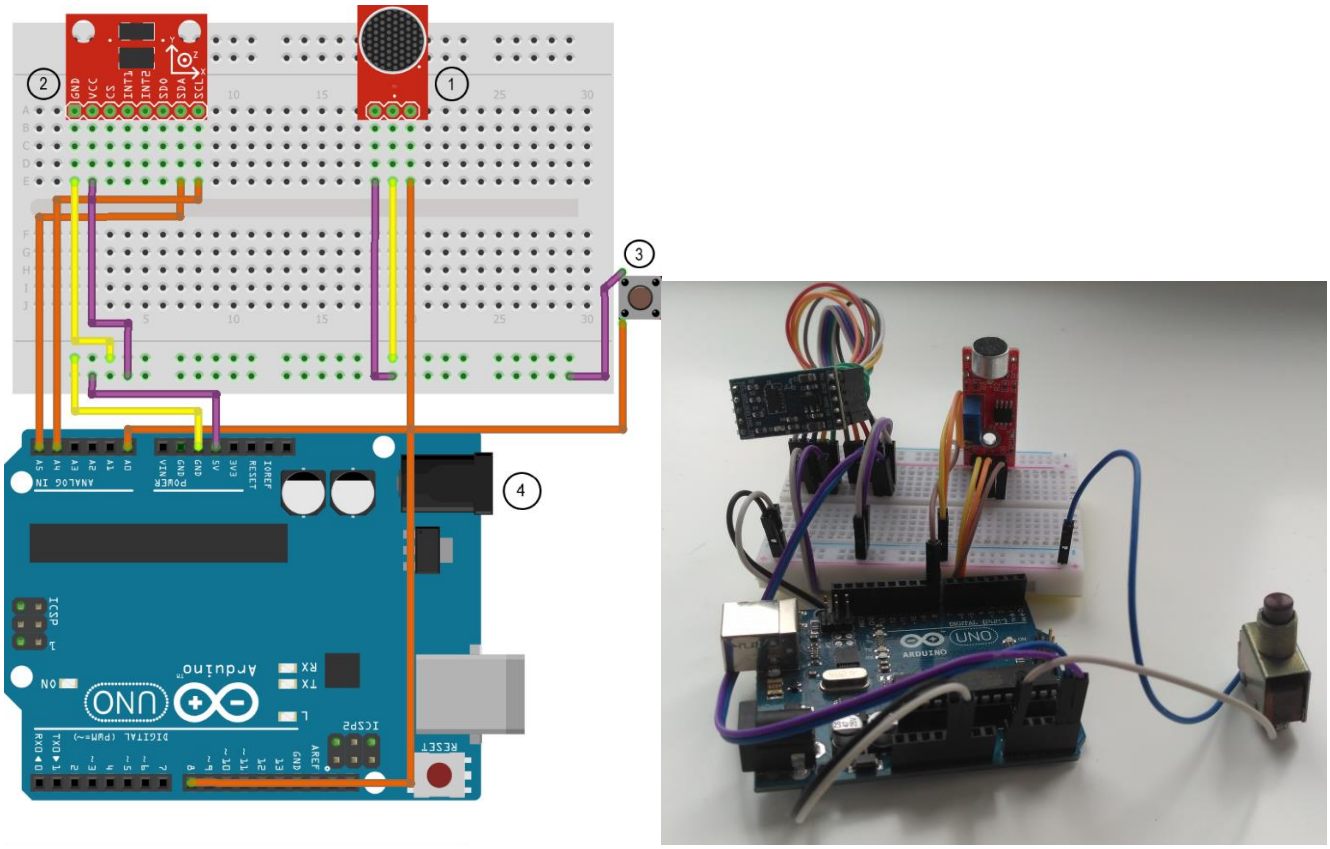

Рис. 4.1. Макетна схема та фізична модель елементів генератора

На схемі наявні такі компоненти:

1. Датчик звуку – схема що містить два головних елементи – потенціометр та мікрофон. Потенціометр використовується для налаштування чутливості вимірів, що також може мати свої переваги при генерації випадкових чисел (наприклад, зміна чутливості в залежності в навколишнього середовища). Мікрофон використовується щоб вимірювати інтенсивність звуку. За рахунок замірів інтенсивності звуку в рівних інтервалах часу можна згенерувати випадкові числа. Даний датчик підключено до цифрового входу Arduino і він повертає 0 або 1 (0 або 5 вольт) в залежності від того, чи перевищує інтенсивність звуку налаштований за допомогою потенціометра рівень.

2. Акселерометр – схема що використовується для вимірювання сили реакції індукованої прискоренням або гравітацією по трьом осям. Її можна використати щоб зібрати випадкові дані коли пристрій змінює орієнтацію або піддається вібрації чи ударам. Для передачі даних до Arduino використовується два аналогових входи на які приходить напруга від 0 до 1023 вольт. Щоб отримати більш «випадкові» заміри, різницю між ними можна збільшити за допомогою множення на константу.

3. Кнопка – тригер що використовується для виклику події генерації випадкового числа. Коли кнопка притиснута, на аналоговий вхід Arduino приходить сигнал. В реальному контролері це б виконувалося іншою схемою, що контролює схему генерації випадкових чисел, але для задач моделювання достатньо і такого підходу.

4. Arduino – головна схема генератора. Вона підключається до джерела струму і забезпечує всі інші елементи необхідним струмом. Arduino також містить програмний код, який виконується генератором та дозволяє поєднувати всі компоненти схеми для виконання спільної задачі створення випадкових чисел.

4.3. Програмне забезпечення генератора випадкових чисел

Схема генератора створена за допомогою Arduino містить всі необхідні компоненти для генерації чисел, але без програмного коду, їх інтеграція між собою не є можливою. Від алгоритму який буде використано також залежить якість генератора. Ці проблеми буде висвітлено в наступному розділі, а зараз розглянемо високорівневий огляд роботи генератора за допомогою блок-схеми (рис. 4.2).

Першим, що може здивувати в блок-схемі, є відсутність кінцевого стану. Це пов’язано зі специфікою Arduino і її подібністю до контролера – програма складається з двох циклів, перший з яких виконується при запуску, а другий є основним циклом і виконується допоки програма не буде перервана.

На самому початку виконання програми генератора виконується ініціалізація, що включає вибір швидкості серійної комунікації та встановлюється

з’єднання з акселерометром. Після цього, дані з окремих сенсорів записуються в окремі ділянки пам’яті, з яких вони потім будуть зчитуватися для генерації випадкової послідовності. Якщо кнопка була натиснута, то виконується генерація випадкового числа, що передається на стандартний вивід. В реальній схемі виведення буде виконуватися до іншого пристрою, а тригером може бути електричний сигнал з третього пристрою, але принцип залишається тим самим.

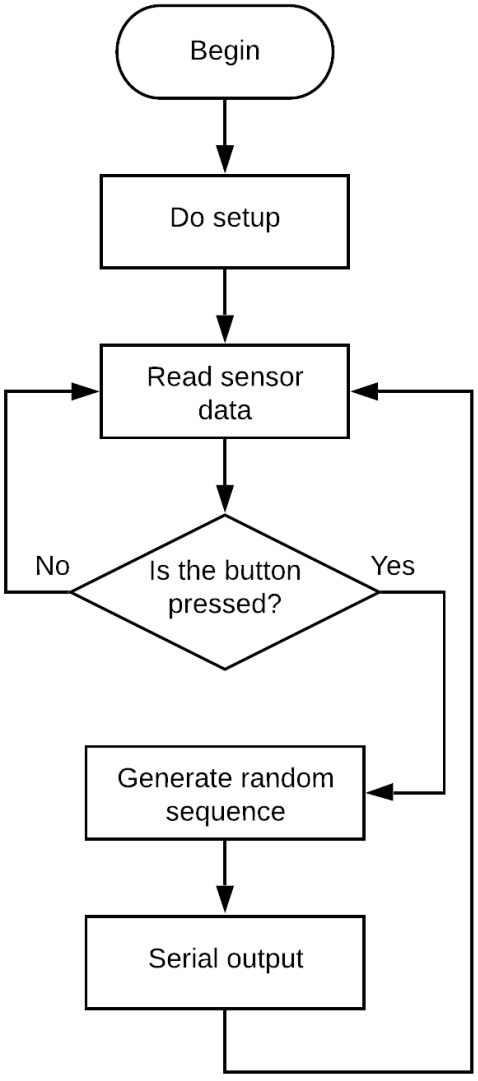

Рис. 4.2. Блок-схема алгоритму програми генератора

РОЗДІЛ 5
ТЕСТУВАННЯ МОДЕЛІ ГЕНЕРАТОРА ВИПАДКОВИХ ЧИСЕЛ

5.1. Генерація випадкових чисел для тестування

Перш ніж генерувати випадкові числа, модель генератора повинна виконувати відповідну програму. Її код наведено в Додатку Е. Модифікувавши програму невеликим чином (для виведення даних), та використовуючи засоби для розробки програм Arduino, можна побачити зміну даних з датчиків в часі (рис. 5.1). Червоний графік відповідає звуку, а помаранчевий, сірий та чорний – осям x, y та z акселерометра відповідно.

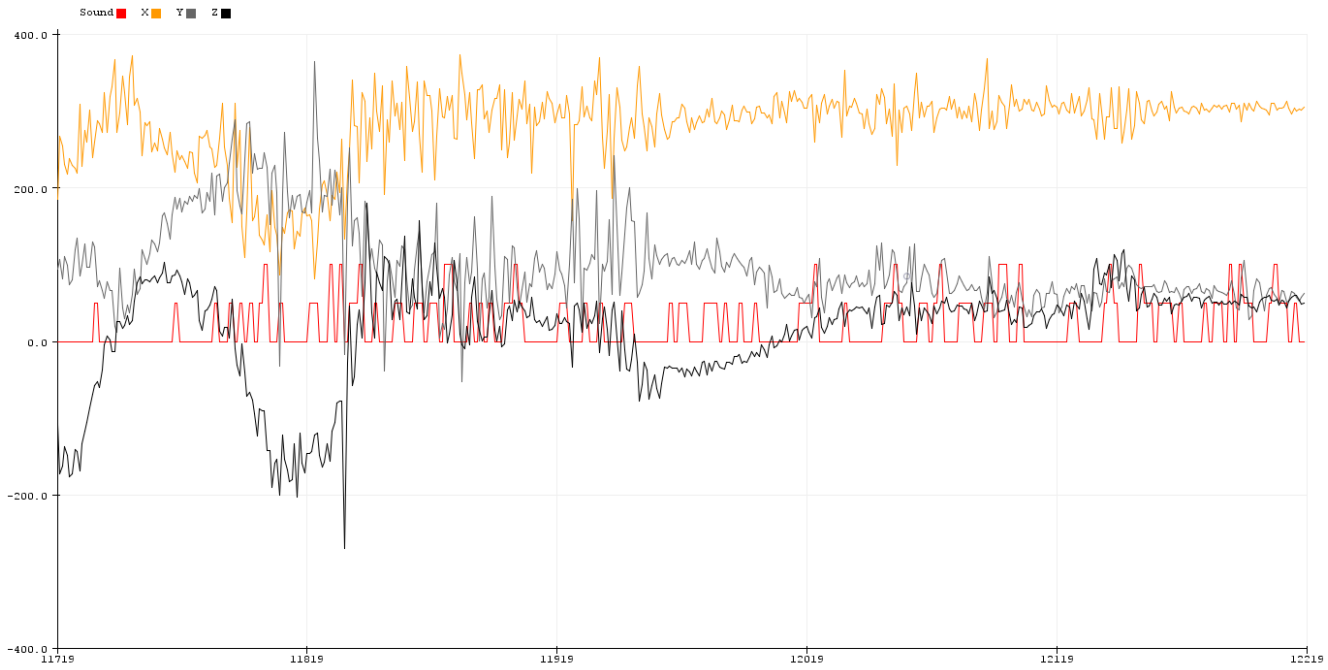

Рис. 5.1. Зміна вимірів з датчиків в часі

Рисунок повною мірою показує переваги АГВЧ – дані є випадковими і вони слабо базуються на попередніх вимірах. Останнім питанням, що може виникнути на шляху до створення якісних випадкових послідовностей, є стійкість алгоритму до відсутності потоку даних або внесення алгоритмом детермінованості в генеровані числа. Ці питання, а також якість створеної моделі можна перевірити за допомогою створеного пакету програм.

Перевірка не була б повною без використання якогось альтернативного методу генерації, адже таким чином можна не тільки показати придатність моделі до реального використання, а і довести що вона показує кращі результати за інші методи генерації. Щоб забезпечити повну перевірку, буде використано те ж середовище Arduino, але генерація випадкових чисел буде проходити за допомогою вбудованої функції random(), що використовує ГПЧ. Для цієї задачі створена наступна програма (рис. 5.2):

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  long value = random(2147483647);
  Serial.println(toBinary(value + value));
  delay(1000);
}

String toBinary(long n) {
  String r;
  while (n != 0) {
    r = (n % 2 == 0 ? "0" : "1") + r;
    n /= 2;
  }
  return r;
}
```

Рис. 5.2. Альтернативна програма з ГПЧ

Варто додати, що обидва генератори створюють послідовності довжини 31 біт, що в більшості мов програмування відповідає типу integer, а в Arduino – типу long. Пам’ять Arduino є доволі обмеженою, що не дозволить генерувати дуже великі послідовності, але це і не є потрібним, адже в експерименті розглядається саме генерація невеликих послідовностей в контексті легковагових генераторів випадкових чисел.

5.2. Тестування, оптимізація та інтерпретація результатів

Тестування згенерованих послідовностей бітів на випадковість буде виконуватися за допомогою методів багатовимірних статистик. Випробування NIST не дадуть якісної оцінки результатів, адже більшість з них розраховані на послідовності, довжина яких більша за 100.

Загальний процес тестування, що буде використано при оцінці генераторів побудовано наступним чином:

1. Запустити програму за допомогою Arduino та згенерувати деяку кількість випадкових послідовностей;

2. Виконати тестування послідовностей з використанням восьми методів багатовимірних статистик наведених в розділі 2 та реалізованих в пакеті програм;

3. Знайти відношення між результатами отриманими на попередньому етапі та максимальними значеннями для тесту і відповідної довжини послідовності. Це значення можна знайти за допомогою визначення найбільш вірогідної кількості входжень шаблонів в послідовність;

4. Зробити висновки про вірогідність послідовності.

Перед тим як розпочати тестування, розглянемо таблицю 5.1, в якій наведено максимальна вірогідність для кожного з тестів багатовимірних статистик.

Таблиця 5.1

Максимальні вірогідності для тестів багатовимірних статистик та довжини послідовності 31

№ Тесту	Найбільша вірогідність
1	0,0699318274855613
2	0,0847552437335252
3	0,0277345534414052
4	0,0368665847927331
5	0,0499579892493784
6	0,0855854991823434
7	0,0172293558716774
8	0,0369482999667525

Щоб оцінити наскільки та чи інша послідовність є випадковою, необхідно знайти відношення результатів відповідного тесту до максимальної вірогідності

цього тесту. Якщо відношення є більшим за 0,8 то послідовність можна вважати випадковою, якщо ні – вона не є випадковою.

Результати тестування 10 послідовностей бітів створених псевдовипадковим генератором на Arduino наведено в таблиці 5.2. Більш повні результати для цього та інших генераторів і алгоритмів наведено в додатку Ж.

Таблиця 5.2

Результати тестування ГПЧ

№ Тесту	1	2	3	4	5	6	7	8
Частина послідовностей що пройшли тест	30%	30%	10%	20%	30%	0%	0%	20%

Результати тестування явно вказують на те, що такі послідовності мають низький рівень випадковості. Оцінити дані можна більш точно, якщо розглянути як показує себе АГВЧ створений з використанням сенсорів (таблиця 5.3). Результати є істотно кращими за ГПЧ і мають загальний позитивний тренд в контексті випадковості.

Таблиця 5.3

Результати тестування АГВЧ

№ Тесту	1	2	3	4	5	6	7	8
Частина послідовностей що пройшли тест	30%	50%	30%	30%	30%	20%	10%	20%

Результати тестування АГВЧ хоч і є позитивними, вказують на те, що цей генератор не створює послідовності належної якості і що його можна потенційно покращити. Однією з проблем зчитування даних з датчиків є те, що дані не змінюються різко і можуть мати загальний тренд. Цю проблему можна вирішити за допомогою внесення додаткової випадковості в процес генерації послідовності, або винесення детермінованості за межі алгоритму. Одним з підходів, що може бути

використано, є Fisher-Yates shuffle [9] – алгоритм, що можна використати для змішування результатів отриманих з датчиків.

Покращену програму АГВЧ з сенсорами можна переглянути в Додатку Е. Результати тестування наведені в таблиці 5.4.

Таблиця 5.4

Результати тестування АГВЧ, що використовує алгоритм Fisher-Yates shuffle

№ Тесту	1	2	3	4	5	6	7	8
Частина послідовностей що пройшли тест	60%	50%	30%	50%	50%	40%	30%	40%

Результати явно вказують на те, що змішування даних отриманих з сенсорів мало позитивний вплив на випадковість послідовностей створюваних генератором.

Наостанок, розглянемо послідовності згенеровані під час того, як генератор та сенсори знаходилися в відносному спокої (таблиця 5.5). Як можна побачити з результатів, коли сенсори не отримують постійних випадкових даних, в послідовностях що створює генератор з’являється висока детермінованість. Відповідно, даний генератор потребує поліпшення на випадок ситуації відносного спокою.

Таблиця 5.5

Результати тестування АГВЧ в стані відносного спокою

№ Тесту	1	2	3	4	5	6	7	8
Частина послідовностей що пройшли тест	20%	20%	10%	10%	10%	20%	10%	10%

РОЗДІЛ 6
ІНСТРУКЦІЯ КОРИСТУВАЧА

6.1. Бібліотека

Бібліотека статистичних тестів поширюється у вигляді файлу формату «.jar». Для використання, її необхідно завантажити і додати до існуючого проекту на мові java. Імпорт бібліотек на Windows можна виконати декількома способами:

1. Якщо не використовується інтегроване середовище розробки, тоді необхідно додати «.jar» як змінну середовища (рис. 6.1).

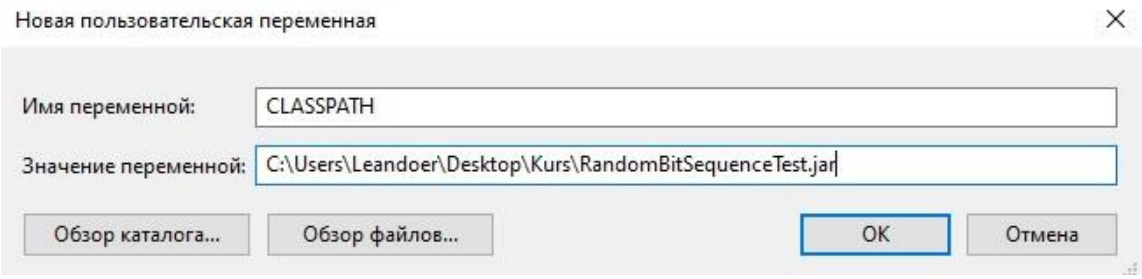

Рис. 6.1. Створення змінної середовища

Як альтернатива, замість зміни середовища, можна вказувати додавати бібліотеку в шлях класу при компіляції та виконанні програми (рис. 6.2).

```
javac -classpath RandomBitSequenceTest.jar Main.java
java -classpath RandomBitSequenceTest.jar Main
```

Рис. 6.2. Компіляція та виконання з бібліотекою

2. Для інтеграції в середовищі розробки IntelliJ IDEA необхідно перейти на вкладку «File > Project Structure > Libraries > New Project Library» та обрати файл бібліотеки з диску. Використання бібліотек з іншими популярними IDE описано в офіційній документації за наступними посиланнями:

– Eclipse – https://help.eclipse.org/2020-03/index.jsp

– NetBeans – https://netbeans.org/kb/74/java/project-setup.html#projects-importing

Щоб додати бібліотеку до проекту з іншого середовища розробки, рекомендується використати його документацію для отримання довідкової інформації.

3. Засоби збірки, також можуть бути використані для імпорту «.jar» файлів. Покрокова інструкція доступна за наступними посиланнями:

– Maven – http://maven.apache.org/guides/mini/guide-3rd-party-jars-local.html

– Gradle – https://discuss.gradle.org/t/installing-3rd-party-jars/14183

Щоб використати бібліотеку в проекті, необхідно імпортувати клас і викликати бажані методи (рис. 6.3).

```
import randombits.nist.NistTest;

public class Main {
    public static void main(String[] args) {
        NistTest.frequencyTest( sequence: "010100111");
    }
}
```

Рис. 6.3. Виклик статистичного тесту

Документація до бібліотеки, а саме: методи бібліотеки, вхідні параметри та їх допустимі значення описано в Додатку А.

6.2. Прикладний програмний Інтерфейс

Доступ до прикладного інтерфейсу можна отримати за рахунок виконання запитів (див. табл. 3.3). Створення HTTP запитів пропонується проводити з використанням мов програмування, однак для тестування API можна використати HTTP клієнт на зразок Postman (https://www.postman.com/). В Додатку Б описано необхідні параметри заголовків, тіла та URL, а також, подано приклади запитів.

6.3. Веб-додаток

При переході на сайт з пошукової системи, користувача буде направлено на головну сторінку (рис. 6.4). Якщо було введено адресу іншої сторінки веб-додатку, або перейдено за посиланням з іншого ресурсу, відкриється сторінка за що знаходиться за конкретною адресою.

Рис. 6.4. Головна сторінка

Головна сторінка надає довідкову інформацію про зміст веб-додатку та корисні посилання. Використавши верхню навігаційну панель, можна перейти на сторінки що відповідають тестам NIST (рис. 6.5) та багатовимірних статистик (рис. 6.6).

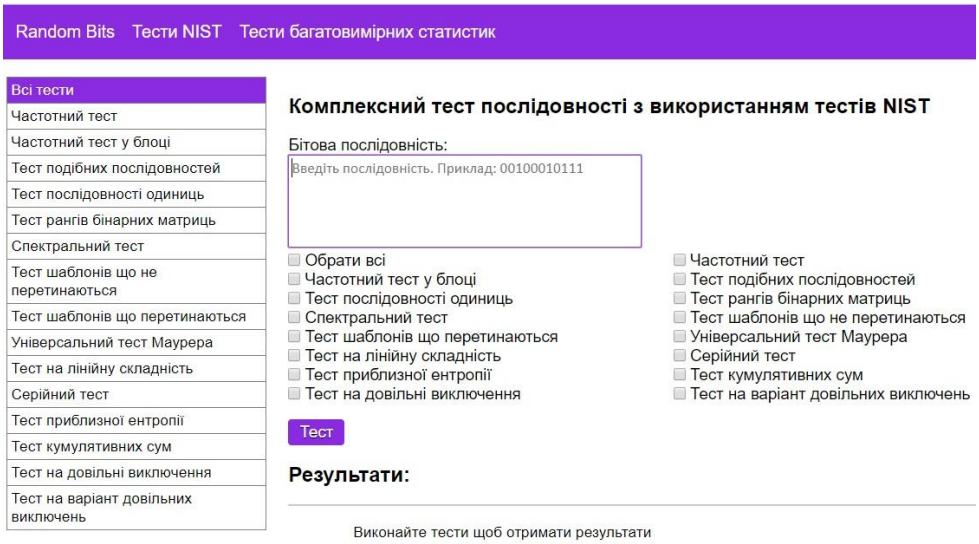

Рис. 6.5. Сторінка NIST

За замовчуванням відкриваються комплексні тести, тобто для одночасного виконання декількох методів. Кожна форма містить елементи управління «Checkbox» для вибору окремих тестів та вибору/зняття всіх одночасно. В текстові поля вводиться послідовність бітів.

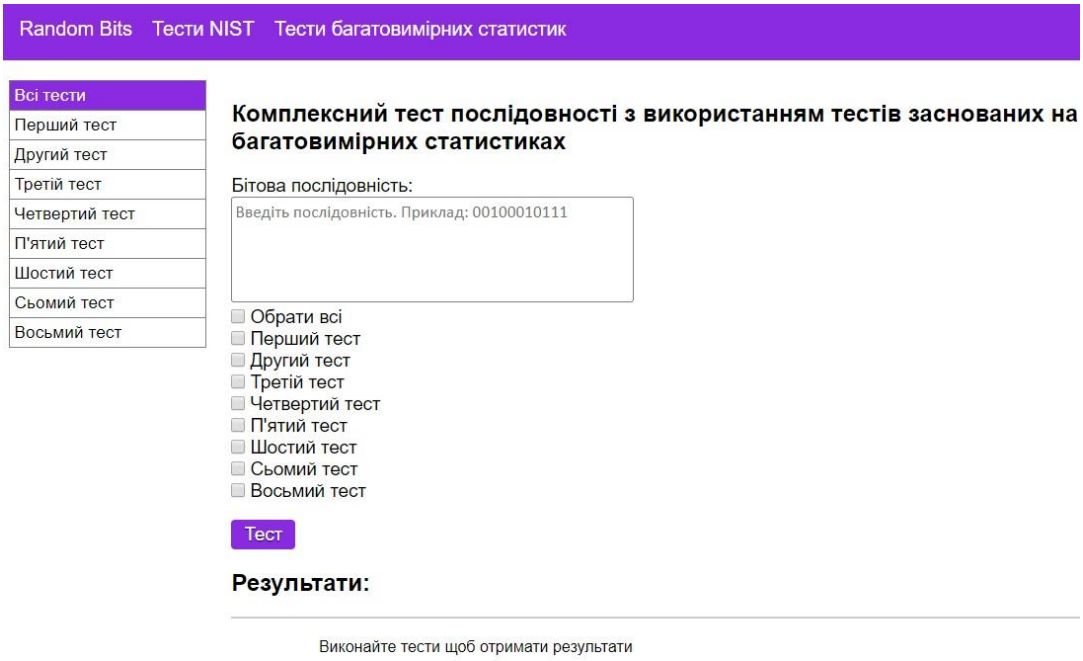

Рис. 6.6. Сторінка тестів багатовимірних статистик

Бокові панелі на обох сторінках містять меню в якому можна обрати окремо один з тестів. На сторінці з методом (рис. 6.7) можна отримати довідкову інформацію і виконати обчислення.

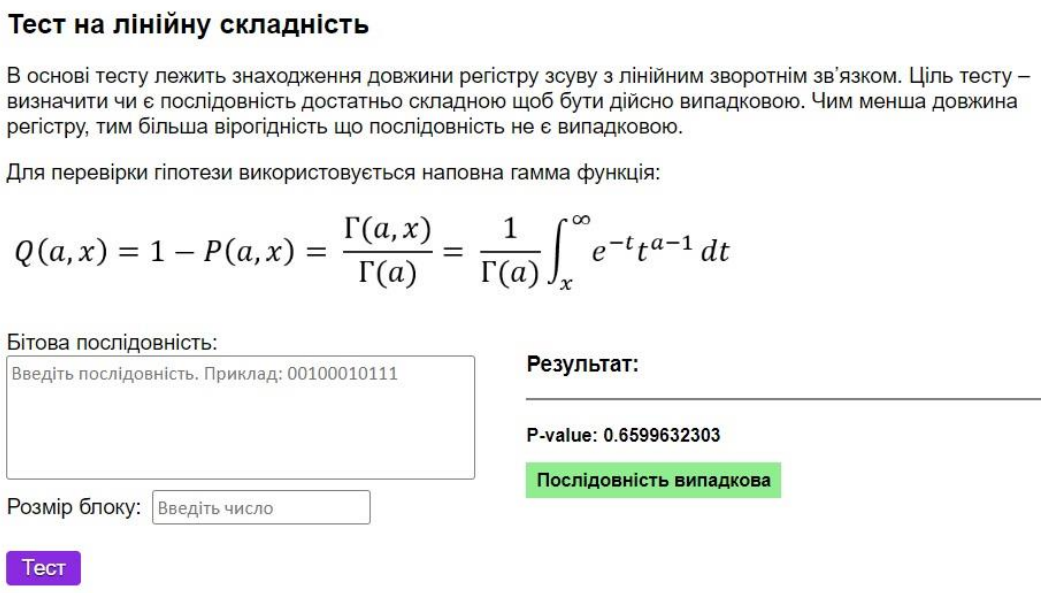

Рис. 6.7. Форма для окремого тесту

Форми містять текстові поля для введення кожного з параметрів тесту, в самих полях є підказки про дані які необхідно вводити. Кнопка «Тест» слугує для виконання методу, а результат буде показано справа. Використання даної кнопки при некоректних даних в полях форми, призводить до появи повідомлення про помилку (рис. 6.8), текст якого завжди відповідає змісту помилки.

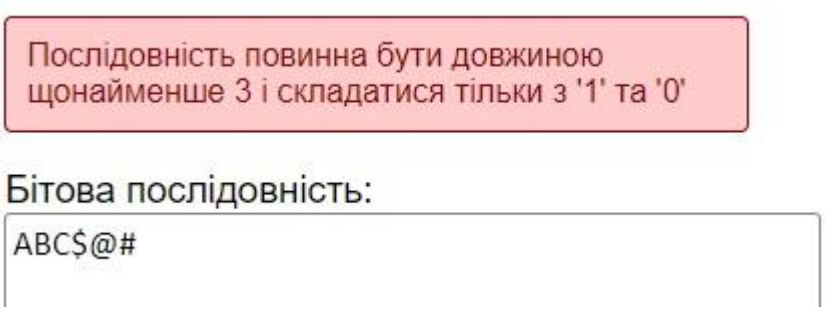

Рис. 6.8. Повідомлення про помилку

Сторінки багатовимірних статистик також містять графіки для презентації результатів тестування. При комплексному тесті, користувачу надається декілька графіків які можна переключати за допомогою кнопок «<» та «>» (рис. 6.9).

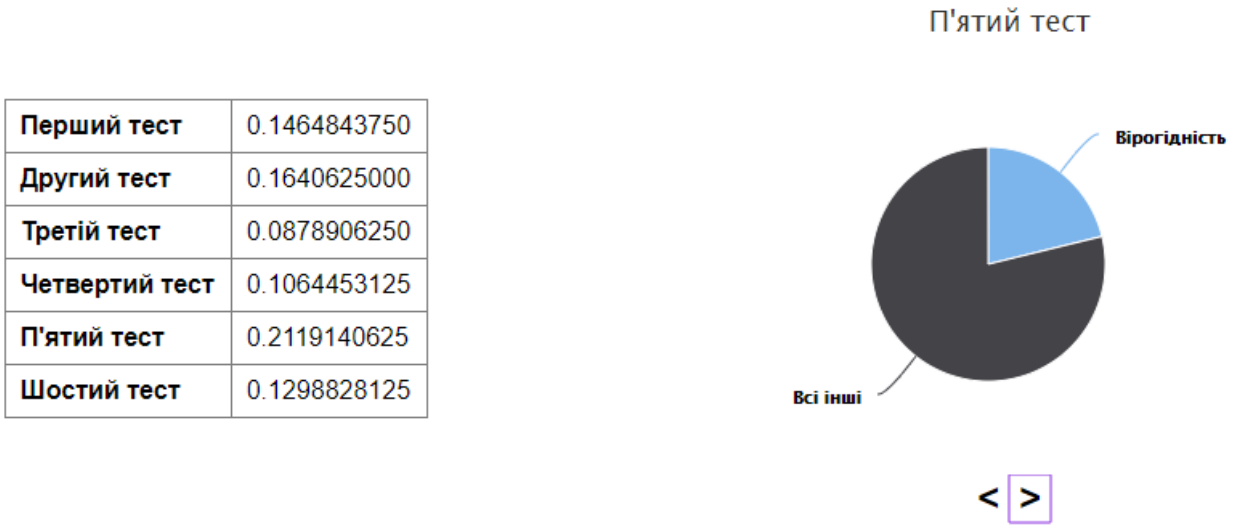

Рис. 6.9. Результати комплексного тесту

ВИСНОВКИ

Дослідження випадкових послідовностей та генераторів випадкових чисел є доволі специфічною задачею, але для її вирішення може бути використаний один або декілька з численних пакетів тестів. Однак, виконаний аналіз вказує на те, що існуючі тести мають низку недоліків, вирішення яких може зменшити передумови до тестування та покращити точність отриманих результатів.

Використання багатовимірних статистик як основи для випробувань, дозволяє краще дослідити послідовність на випадковість, за рахунок оцінки одночасно декількох характеристик послідовності. Тести багатовимірних статистик засновані на дослідженні входжень шаблонів в послідовність і допомагають виявляти приховані залежності між даними та неякісні генератори. Головною перевагою цих тестів є їх ефективність на послідовностях короткої довжини, тому вони вирішують одну з проблем існуючих тестів, полегшуючи передумови до випробувань.

Пакет програм, що створено з 15 тестів NIST та 8 тестів заснованих на використанні багатовимірних статистик, придатний до тестування набагато більшої кількості послідовностей та генераторів в порівнянні з існуючими пакетами тестів. Додатковою перевагою є те, що він створений з урахуванням потреб різних користувачів і містить такі програми:

- Бібліотека на мові Java, що включає 15 тестів NIST та 8 тестів багатовимірних статистик;

- Прикладний програмний інтерфейс, що надає можливість виконувати тести за допомогою HTTP запитів;

- Веб-додаток, який надає користувацький інтерфейс для звичайних користувачів.

Фізична модель IoT генератора представлена в роботі, на своєму прикладі надає широкий огляд факторів та обмежень, що виникають під час проектування генераторів. Процес тестування та оптимізації генератора з використанням тестів багатовимірних статистик ілюструє придатність пакету програм до використання і

його інтегральну роль в створенні якісного генератора випадкових чисел, в особливості для використання в IoT пристроях.

Пакет тестів і пакет програм в цілому, рекомендовано до використання при дослідженні генераторів випадкових чисел та послідовностей створених ними на випадковість. Вони можуть бути застосовані в одні з наступних областей:

– Наукові дослідження – встановлення залежності між будь-якими експериментальними даними, розробка ГПЧ та АГВЧ (як для IoT пристроїв, так і для загального використання), створення нових методів перевірки послідовності на випадковість;

– Криптографія – перевірка послідовностей згенерованих ГПЧ та АГВЧ, дослідження алгоритмів шифрування;

– Розробка та супровід програмних продуктів – тестування ефективності алгоритмів та систем заснованих на випадковості, перевірка криптографічних засобів систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications [Електронний ресурс] / [A. Rukhin, J. Soto, J. Nechvatal та ін.] // National Institute of Standards and Technology. – 2010. – Режим доступу до ресурсу: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf.

2. DIEHARD Statistical Tests [Електронний ресурс] – Режим доступу до ресурсу: https://stat.fsu.edu/pub/diehard/.

3. Diehard tests [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Diehard_tests.

4. TestU01: A software library in ANSI C for empirical testing of random number generators [Електронний ресурс] // Department d'Informatique et de Recherche Operationnelle, University of Montreal. – 2013. – Режим доступу до ресурсу: http://simul.iro.umontreal.ca/testu01/guideshorttestu01.pdf.

5. Masol V. I. Statistical analysis of local sections of bits sequences / V. I. Masol, S. V. Popereshnyak. // Journal of Automation and Information Sciences. – 2019. – №51. – С. 31–45.

6. Popereshnyak S. The Technique for Testing Short Sequences as a Component of Cryptography on the Internet of Things / Svitlana Popereshnyak. // CEUR Workshop Proceedings. – 2019. – С. 138–148

7. Popereshnyak S. One Way of Testing of Pseudorandom Sequence of Small Length Using Multidimensional Statistics / Svitlana Popereshnyak. // IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology. – 2019. – С. 649–654.

8. Ullah I. Entropy as a Service: A Lightweight Random Number Generator for Decentralized IoT Applications [Електронний ресурс] / I. Ullah, N. Meratnia, P. Havinga // IEEE. – 2020. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/abstract/document/9156205.

9. Fisher–Yates shuffle [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle.

ДОДАТКИ

Додаток А

Документація до методів бібліотеки

Таблиця А.1

Методи бібліотеки та вхідні параметри

Інтерфейс	**Метод**	
	Вхідні параметри	**Область значень**
NisrTest	TestResult frequencyTest()	
	sequence (string) – бітова послідовність	Довжина послідовності має бути більше ніж 3 і складатися тільки з «0» та «1»
	TestResult blockFrequencyTest()	
	sequence (string) – бітова послідовність	Довжина послідовності має бути більше ніж 3 і складатися тільки з «0» та «1»
	blockSize (int) – розмір блоку для тесту	Довжина блоку має бути менше ніж довжина послідовності та більше ніж 1
	TestResult runsTest()	
	sequence (string) – бітова послідовність	Довжина послідовності має бути більше ніж 3 і складатися тільки з «0» та «1»
	TestResult longestRunOfOnesTest()	
	sequence (string) – бітова послідовність	Довжина послідовності має бути більше ніж 3 і складатися тільки з «0» та «1»
	TestResult binaryMetrixRankTest()	
	sequence (string) – бітова послідовність	Довжина послідовності має бути більше ніж 3 і складатися тільки з «0» та «1»
	matrixSize (int) – розмір матриць тесту	Розмір матриці повинен бути більше ніж 1 та менше ніж корінь послідовності ($matrixSize^2$ < довжина послідовності)

Продовження табл. А.1

Інтерфейс	**Метод**	
	Вхідні параметри	**Область значень**
	TestResult spectralTest()	
	sequence (string) – бітова послідовність	Довжина послідовності має бути більше ніж 3 і складатися тільки з «0» та «1»
NisrTest	TestResult nonOverlappingTemplateTest()	
	sequence (string) – бітова послідовність	Довжина послідовності має бути більше ніж 3 і складатися тільки з «0» та «1»
	blockSize (int) – розмір блоку для тесту	Довжина блоку має бути менше ніж довжина послідовності та більше ніж 1
	template (string) – бітовий шаблон	Довжина шаблону має бути більше 1 та складатися тільки з «0» та «1»
	TestResult overlappingTemplateTest()	
	sequence (string) – бітова послідовність	Довжина послідовності має бути більше ніж 3 і складатися тільки з «0» та «1»
	blockSize (int) – розмір блоку для тесту	Довжина блоку має бути менше ніж довжина послідовності та більше ніж 1
	template (string) – бітовий шаблон	Довжина шаблону має бути більше 1 та складатися тільки з «0» та «1»
	TestResult maurersTest()	
	sequence (string) – бітова послідовність	Довжина послідовності має бути більше ніж 3 і складатися тільки з «0» та «1»
	blockSize (int) – розмір блоку для тесту	Довжина блоку має бути менше ніж довжина послідовності в 10 разів або менша ніж 16, та більше ніж 1
	blocksInInitSegment (int) – довжина початкового сегменту	Довжина сегменту повинна бути більше 1, та не перевищувати половину послідовності при добутку з blockSize

Продовження табл. А.1

Інтерфейс	**Метод**	
	Вхідні параметри	**Область значень**
NistTest	TestResult linearComplexityTest()	
	sequence (string) – бітова послідовність	Довжина послідовності має бути більше ніж 3 і складатися тільки з «0» та «1»
	blockSize (int) – розмір блоку для тесту	Довжина блоку має бути менше ніж довжина послідовності та більше ніж 1
	List<TestResult> serialTest()	
	sequence (string) – бітова послідовність	Довжина послідовності має бути більше ніж 3 і складатися тільки з «0» та «1»
	blockSize (int) – розмір блоку для тесту	Довжина блоку має бути менше ніж 16 та більше ніж 3
	TestResult entropyTest()	
	sequence (string) – бітова послідовність	Довжина послідовності має бути більше ніж 3 і складатися тільки з «0» та «1»
	blockSize (int) – розмір блоку для тесту	Довжина блоку має бути менше ніж 15 та більше ніж 2
	List<TestResult> cumulativeSumsTest()	
	sequence (string) – бітова послідовність	Довжина послідовності має бути більше ніж 3 і складатися тільки з «0» та «1»
	List<TestResult> randomExcursionsTest()	
	sequence (string) – бітова послідовність	Довжина послідовності має бути більше ніж 3 і складатися тільки з «0» та «1»
	List<TestResult> randomExcursionsVariantTest()	
	sequence (string) – бітова послідовність	Довжина послідовності має бути більше ніж 3 і складатися тільки з «0» та «1»

Продовження табл. А.1

Інтерфейс	**Метод**	
	Вхідні параметри	**Область значень**
MdTest	double one(); double two(); double three(); double four(); double five(); double six(); double seven(); double eight().	
	sequence (string) – бітова послідовність	Довжина послідовності має бути більше ніж 3 і складатися тільки з «0» та «1»

Додаток Б

Документація до методів API за доступними URL

Таблиця Б.1

Запит окремого тесту NIST

Поле	**Значення**
Метод HTTP	POST
Відносний URL	/nist_test/{test_name}
Параметри заголовку запитів	Content-Type: application/json
Очікуваний формат тіла запиту	{ <назва параметру 1>: <значення 1>, <назва параметру 2>: <значення 1>, …}

Приклад запиту	URL: /nist_test/block_frequency
	Тіло запиту:
	{
	«sequence»: «1010101010011»,
	«blockSize»: «2»
	}

Приклад відповіді	{
	«pValue»: 0.1010198178,
	«isSequenceRandom»: True
	}

Таблиця Б.2

Запит для декількох тестів NIST

Поле	**Значення**
Метод HTTP	POST
Відносний URL	/nist_test
Параметри заголовку запитів	Content-Type: application/json

Очікуваний формат тіла запиту	{ <назва тесту 1>: {
	<назва параметру 1>: <значення 1>,
	<назва параметру 2>: <значення 1>
	…},
	<назва тесту 2>: {
	<назва параметру 1>: <значення 1>,
	<назва параметру 2>: <значення 1>,
	…}, …}

Приклад запиту	URL: /nist_test
	Тіло запиту:
	{«binary_matrix»: {
	«sequence»: «10101010101010111»,
	«matrixSize»: «2»},
	«excursion_variant»: {
	«sequence»: «10101010101010111»}}

Приклад відповіді	{«binary_matrix»: {
	«pValue»: 0.0331935,
	«isSequenceRandom»: false},
	«excursion_variant»: {
	[{«pValue»: 0.789213,
	«isSequenceRandom»: True
	},{«pValue»: 3.145123E-8,
	«isSequenceRandom»: false}, …]}

Таблиця Б.3

Запит для окремого тесту багатовимірних статистик

Поле	**Значення**
Метод HTTP	POST
Відносний URL	/md_test/{test_name}
Параметри заголовку запитів	Content-Type: application/json

Очікуваний формат тіла запиту	{
	«sequence»: <значення 1>
	}

Приклад запиту	URL: /md_test/two
	Тіло запиту:
	{
	«sequence»: «1010101010011»
	}

Приклад відповіді	{
	«value»: 0.023791
	}

Таблиця Б.4

Запит для комплексного тесту багатовимірних статистик

Поле	**Значення**
Метод HTTP	POST
Відносний URL	/md_test?tests=[test1, test2…]
Параметри заголовку запитів	Content-Type: application/json

Очікуваний формат тіла запиту	{
	«sequence»: <значення 1>
	}

Приклад запиту	URL: /md_test?tests=[three, seven]
	Тіло запиту:
	{
	«sequence»: «1010101010011»
	}

Приклад відповіді	{
	«three»: {
	«value»: 0.031113
	},
	«seven»: {
	«value»: 0.2875
	}
	}

Додаток В

Повні діаграми класів

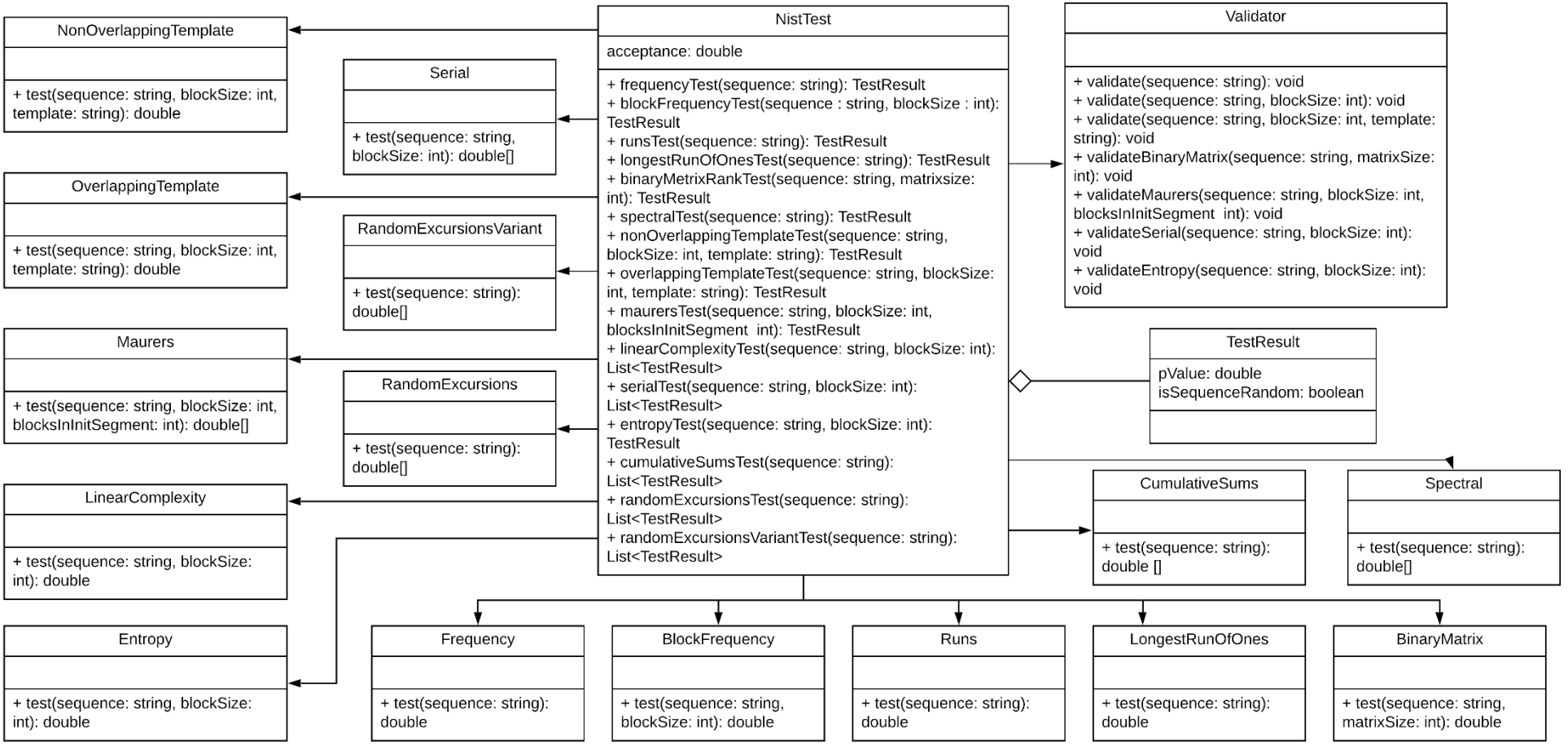

Рис. В.1 Діаграма класів для пакету з тестами NIST

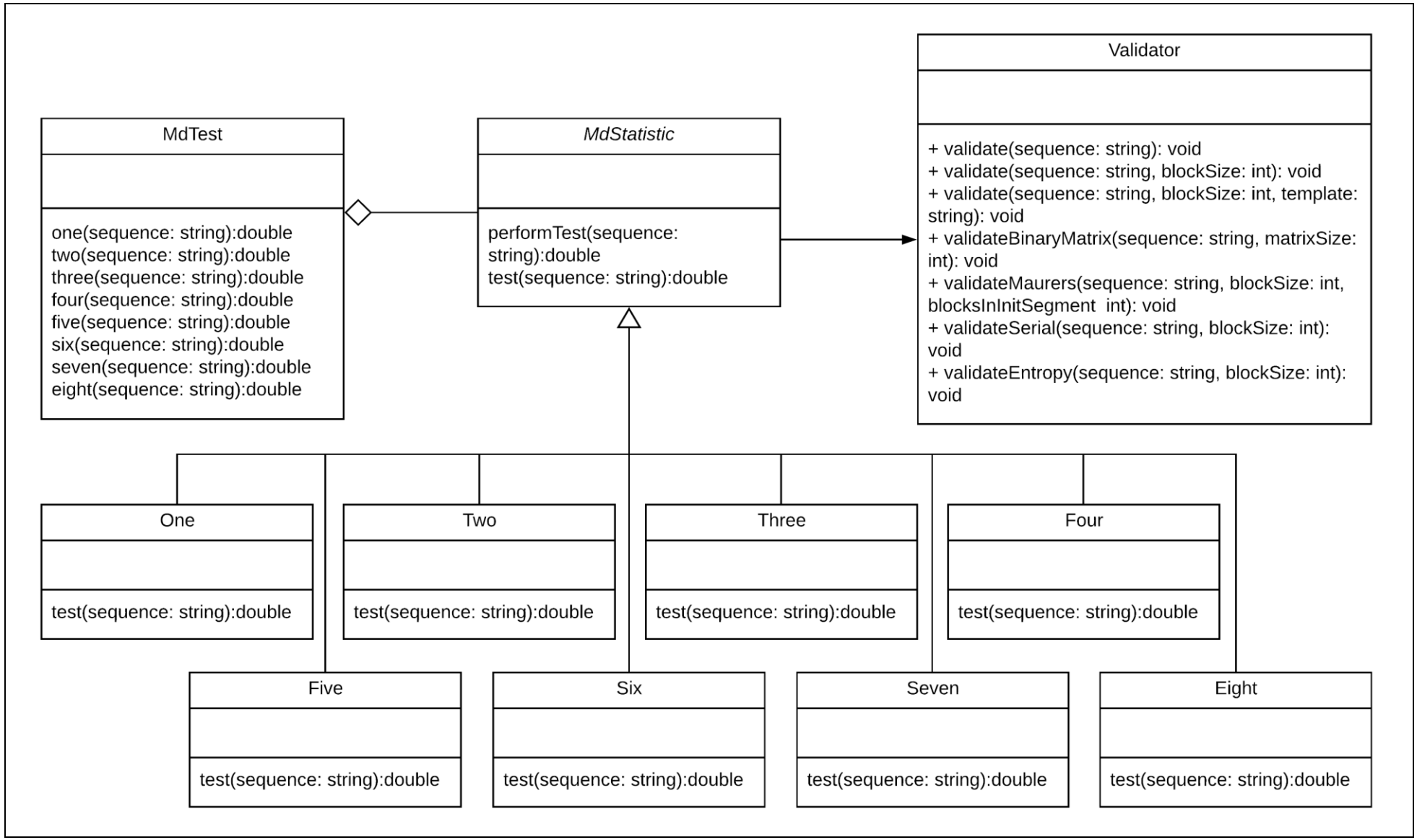

Рис. В.2 Діаграма класів для пакету з багатовимірними статистиками

Додаток Д

Код деяких модулів програми

Клас з тестом приблизної ентропії - Entropy.java

```
public class Entropy {

    public static double test(String sequence, int blockSize) {
        String[] sequences = {sequence + sequence.substring(0, blockSize - 1), sequence +
sequence.substring(0, blockSize)};
        int n = sequence.length();
        double[] sums = new double[2];
        for (int i = 0; i <= 1; i++) {
            String[] temp = TemplateUtils.getTemplates(blockSize + i);
            double[] frequencies = new double[temp.length];
            for (int j = 0; j < temp.length; j++) {
                int k = blockSize + i;
                while (k <= sequences[i].length()) {
                    if (sequences[i].substring(k - blockSize - i, k).equals(temp[j])) {
                        frequencies[j]++;
                    }
                    k++;}}
            sums[i] = Arrays.stream(frequencies)
                    .map(x -> x / n)
                    .map(x -> {
                        if (x != 0) {
                            return x * (Math.log(x));
                        } else {
                            return x;
                        }
                    }).sum();
        }
        double chiSquared = 2 * n * (Math.log(2) - (sums[0] - sums[1]));
        return GammaUtils.gammaFunction(Math.pow(2, blockSize - 1), chiSquared / 2);
    }
}
```

Клас четвертого методу багатовимірних статистик Four.java

```
public class Four extends MdStatistic {

    //{k1 = n(tt*), k2 = n(t1t) + n(t0t)}
    protected double test(String sequence) {
        int k1 = TemplateUtils.countOverlapping(sequence, "01");
        int k2 = TemplateUtils.countOverlapping(sequence, "010") +
                TemplateUtils.countOverlapping(sequence, "000");
        double total = 0;
        for (int m1 = k1; m1 <= sequence.length() - k1; m1++) {
            int m0 = sequence.length() - m1;
            for (int delta0 = 0; delta0 <= k2; delta0++) {
                for (int delta1 = 0; delta1 <= k2; delta1++) {
                    if (delta0 + delta1 == k2) {
                        for (int i = k1; i <= k1 + 1; i++) {
                            double c = m1 >= 2 ? 1 : 0;
                            total += FunctionUtils.binomialCoefficient(i - 1, delta1) *
                                    FunctionUtils.binomialCoefficient(i, delta0 - m1 + 2 *
i) *
                                    FunctionUtils.binomialCoefficient(m0 - i + 1, k1 -
delta1) *
                                    FunctionUtils.functionZ(m1 - i, m1 - i - delta0) * c;
```

```
}}}}
            double z = (m1 == 0 ? 1 : 0) * (!(k1 + k2 >= 1) ? 1 : 0) +
                    (m1 == 1 ? 1 : 0) * (!(k2 >= 1) ? 1 : 0) * (!(k2 == 0 && k1 >= 2) ? 1
: 0) * ((k1 == 0 && k2 == 0 ? 1 : 0) + (sequence.length() - 1) * (k1 == 1 && k2 == 0 ? 1 :
0));
            total += z;
        }
        return 1 / Math.pow(2, sequence.length()) * total;}}
```

Частина класу NistTest.java

```
public class NistTest {
    public final static double acceptance = 0.01;
    public static TestResult frequencyTest(String sequence) {
        Validator.validate(sequence);
        double pValue = Frequency.test(sequence);
        return new TestResult(pValue, pValue >= acceptance);
    }

    public static TestResult blockFrequencyTest(String sequence, int blockSize) {
        Validator.validate(sequence, blockSize);
        double pValue = BlockFrequency.test(sequence, blockSize);
        return new TestResult(pValue, pValue >= acceptance);
    }

    public static List<TestResult> cumulativeSumsTest(String sequence) {
        Validator.validate(sequence);
        double[] pValues = CumulativeSums.test(sequence);
        return Arrays.stream(pValues).mapToObj(item -> new TestResult(item, item >=
acceptance)).collect(Collectors.toList());}
```

. . . . . .

Клас MdStatistic.java

```
public abstract class MdStatistic {

    public double performTest(String sequence) {
        Validator.validate(sequence);
        return test(sequence);
    }

    protected abstract double test(String sequence);
}
```

Частина класу MdTest.java

```
public class MdTest {

    public static double one(String sequence) {
        MdStatistic statistic = new One();
        return statistic.performTest(sequence);
    }

    public static double two(String sequence) {
        MdStatistic statistic = new Two();
        return statistic.performTest(sequence);
    }
```

. . . . . .

Клас для результатів тестів NIST – TestResult.java

```
public class TestResult {
    private double pValue;
    private boolean isSequenceRandom;

    public TestResult(double pValue, boolean result) {
        this.pValue = pValue;
        this.isSequenceRandom = result;
    }

    public double getpValue() {
        return pValue;
    }

    public boolean isSequenceRandom() {
        return isSequenceRandom;
    }

    @Override
    public String toString() {
        return "{p_value: " + this.pValue + ", random: " + this.isSequenceRandom + " }";
    }
}
```

Частина класу обробки виключень при запитах ErrorHandler.java

```
@ControllerAdvice
public class ErrorHandler extends ResponseEntityExceptionHandler {
    @Override
    protected ResponseEntity<Object>
handleHttpRequestMethodNotSupported(HttpRequestMethodNotSupportedException ex,
HttpHeaders headers, HttpStatus status,WebRequest request) {
        ErrorEntity errorEntity = new ErrorEntity();
        errorEntity.setTimestamp(ZonedDateTime.now(ZoneId.of("UTC")));
        errorEntity.setStatus(HttpStatus.METHOD_NOT_ALLOWED.value());
        errorEntity.setError(HttpStatus.METHOD_NOT_ALLOWED.getReasonPhrase());
        errorEntity.setMessage(((ServletWebRequest) request).getRequest().getMethod() +
                " method is not supported by this URL.");
        errorEntity.setPath(((ServletWebRequest) request).getRequest().getRequestURI());
        String methods = ex.getSupportedHttpMethods().toString();
        headers.add("Allow", methods.substring(1, methods.length() - 1));
        return new ResponseEntity<>(errorEntity, headers, HttpStatus.METHOD_NOT_ALLOWED);}
```

. . . . . .

Клас сутності помилок ErrorEntity.java

```
public class ErrorEntity {
    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd'T'HH:mm:ss.SSSZ")
    private
    ZonedDateTime timestamp;
    private int status;
    private String error;
    private String message;
    private String path;

    public ErrorEntity(ZonedDateTime dateTime, int status, String error, String message,
String path) {
        this.setTimestamp(dateTime);
```

```
        this.setStatus(status);
        this.setError(error);
        this.setMessage(message);
        this.setPath(path);
    }
}
```

Частина фабрики з використанням типу, що перераховується NistFactory.java

```
public enum NistFactory {
    FREQUENCY {
        @Override
        @SuppressWarnings("unchecked")
        public <T> T test(Map<String, String> testParams) {
            return (T) NistTest.frequencyTest(testParams.get("sequence"));
        }
    },

    BLOCK_FREQUENCY {
        @Override
        @SuppressWarnings("unchecked")
        public <T> T test(Map<String, String> testParams) {
            return (T) NistTest.blockFrequencyTest(testParams.get("sequence"),
                    Integer.parseInt(testParams.get("blockSize")));
        }
    },
    BINARY_MATRIX {
        @Override
        @SuppressWarnings("unchecked")
        public <T> T test(Map<String, String> testParams) {
            return (T) NistTest.binaryMatrixRankTest(testParams.get("sequence"),
                    Integer.parseInt(testParams.get("matrixSize")));
        }
    }
. . . . . .
```

Контролер для запитів на тести NIST NistController.java

```
@RestController
@RequestMapping("/nist_test")
public class NistController {

    NistService nistService;

    @Autowired
    public NistController(NistService nistService) {
        this.nistService = nistService;
    }

    @PostMapping
    Map<String, Object> testSequenceAll(@RequestBody Map<String, Map<String, String>>
params) {
        return nistService.computeNistStatisticForAll(params);
    }

    @PostMapping("/{test}")
    Object testSequence(@PathVariable("test") NistFactory method, @RequestBody Map<String,
String> params) {
        return nistService.computeNistStatistic(method, params);}}
```

Клас з бізнес логікою тестів NIST NistServiceImpl.java

```
@Service
public class NistServiceImpl implements NistService {

    @Override
    public Object computeNistStatistic(NistFactory method, Map<String, String>
testParams){ return method.getResult(testParams); }

    @Override
    public Map<String, Object> computeNistStatisticForAll(Map<String, Map<String, String>>
testParams) {
        Map<String, Object> results = new LinkedHashMap<>();
        for (Map.Entry<String, Map<String, String>> map: testParams.entrySet()){
            NistFactory method = NistFactory.valueOf(map.getKey().toUpperCase());
            results.put(method.toString().toLowerCase(),
method.getResult(map.getValue()));
        }
        return results;}}
```

Клас з бізнес логікою тестів багатовимірних статистик MdTest.java

```
@Service
public class MdServiceImpl implements MdService {

    @Override
    public MdResult computeMdStatistic(MdFactory method, Map<String, String> testParams) {
        return method.getResult(testParams);
    }

    @Override
    public Map<String, MdResult> computeMdStatisticForAll(List<MdFactory> methods,
Map<String, String> map) {
        Map<String, MdResult> results = new LinkedHashMap<>();
        for (MdFactory method: methods){
            results.put(method.toString().toLowerCase(), method.getResult(map));
        }
        return results;
    }
}
```

Деякі інтеграційні тести з класів NistApiTest.java та MdApiTest.java

```
@Test
public void shouldReturnUnprocessableEntityOnBadTestParameters() throws Exception {
    this.mockMvc.perform(post("/md_test/two")
            .contentType(MediaType.APPLICATION_JSON)
            .content("{\"sequence\": \"0\"}")
    )
            .andExpect(status().is(422))
            .andExpect(jsonPath("$.message")
                    .value("Illegal sequence length ( <3): 1"));

    this.mockMvc.perform(post("/md_test/three")
            .contentType(MediaType.APPLICATION_JSON)
            .content("{\"sequence\": \"abcdef\"}")
    )
            .andExpect(status().is(422))
            .andExpect(jsonPath("$.message")
                    .value("Sequence should contain only ones (1) and zeroes (0)."));
}
```

```
@Test
public void shouldReturnResultWhenGoodParameters() throws Exception {
    this.mockMvc.perform(post("/md_test/one")
            .contentType(MediaType.APPLICATION_JSON)
            .content("{\"sequence\":
\"11001001000011111101010100010001000010110001010001011100 0\"}"))
            .andExpect(status().isOk())
            .andExpect(content()
                    .json("{\"value\": 0.012888100881170567}"));
}
```

Деякі з модульних тестів для серверу та бібліотеки

```
@Test
public void shouldReturnHttpBadRequestIfNoSuchTest() throws Exception {
    this.mockMvc.perform(post("/nist_test/ffrequency"))
            .andExpect(status().is(400))
            .andExpect(jsonPath("$.message")
                    .value("Url parameter should be one of: frequency, block_frequency, "
+
                            "binary_matrix, non_overlapping_template,
overlapping_template, spectral, " +
                            "linear_complexity, longest_run_of_ones, maurers,
cumulative_sums, " +
                            "random_excursions, excursion_variant, runs, serial,
entropy."));

    this.mockMvc.perform(post("/md_test/abc123xyz"))
            .andExpect(status().is(400))
            .andExpect(jsonPath("$.message")
                    .value("Url parameter should be one of: " +
                            "one, two, three, four, five, six, seven, eight."));
}


@Test
public void shouldReturnMethodNotAllowed() throws Exception {
    this.mockMvc.perform(get("/nist_test/cumulative_sums"))
            .andExpect(status().is(405))
            .andExpect(jsonPath("$.message")
                    .value("GET method is not supported by this URL."));
    this.mockMvc.perform(get("/md_test/eight"))
            .andExpect(status().is(405))
            .andExpect(jsonPath("$.message")
                    .value("GET method is not supported by this URL."));
}

@Test
public void testEight() {
    Random r = new Random();
    for (int i = 0; i < 1000; i++) {
        String temp = Integer.toBinaryString(r.nextInt(65533) + 4);//2^16
        assertEquals(TestUtils.impericalTwoFromTwo(temp, "00", "101"),
                MdTest.eight(temp), 0.001);
    }
}
```

Компонент сторінки методу NIST – Maurers.js

```
class Maurers extends NistTest {
    validate(data) {
        return validateMaurers(data);}
```

```
    render() {
        return (
            <div>
                <h2>Універсальний тест Маурера</h2>
                <p>
                    Ціль тесту - перевірити кількість бітів між подібними шаблонами. На
основі цього, робиться висновок про можливий рівень компресії послідовності. Послідовність
що можна сильно стиснути не є випадковою.
<p>Для перевірки гіпотези використовується доповнююча функція помилок:</p>
                    <div className={"nist-formula"}>
                        <img src={erfc2}/>
                    </div>
                </p>

                <div className={"test-form-nist"}>
                    <InputForm state={{sequence: "",blockSize: "", blocksInInitSegment:
""}}
                               name={{sequence: "Бітова послідовність:", blockSize:
"Розмір блоку:", blocksInInitSegment:"Кількість блоків: "}}
url={"maurers"}
                               update={this.updateResult}
                               validator={this.validate}
                               placeholder={{blockSize: "Введіть число",
blocksInInitSegment: "Введіть число"}}/>
                    {this.state.result ? <ResultDisplay result={this.state.result}/>
:<div/>}
                </div>
            </div>
        );
    }
}
export default Maurers;
```

Компонент форми для тестів NIST InputForm.js

```
class InputForm extends Component {
    constructor(props) {
        super(props);
        this.state = {
            params: this.props.state,
            error: false
        }
    }


    normalize = () =>{
        const params = this.state.params
        Object.keys(params).map((item) =>{
            params[item] = params[item].replace(/\s/g, "")
            return params;
        })
        return params
    }

    onClick = () =>{
        const params = this.normalize();
        this.setState({error:this.props.validator(this.state.params)})
        if(!this.props.validator(this.state.params)){

        fetch(`http://localhost:8080/nist_test/${this.props.url}`, {
            method:"POST",
```

```
            mode: "cors",
            headers: {
                'Content-Type': "application/json"
            },
            body: JSON.stringify(params)
        }).then((response) => response.json())
            .then((data) => {this.props.update(data, params)
                this.setState({data})})
            .catch(
            () => {throw `API error on ${this.props.url}`}
        )
    } else { this.props.update(0,0); }}

    onChange = (event) =>{
        const {name, value} = event.target;
        this.setState(prevState =>{
            let params = Object.assign({}, prevState.params)
            params[[name]] = value;
            return {params};
        });

    }

    render() {
        return (
            <div>
                {this.state.error &&
                <><div className={"error"}>{this.state.error}</div><br/></>}
                <label>{this.props.name["sequence"]}</label>
                <br />
                <textarea name={"sequence"} value={this.state.params.sequence}
onChange={this.onChange} placeholder={"Введіть послідовність. Приклад: 00100010111"}/>
                <br/>
                {Object.keys(this.state.params).map( (key) =>{
                        if(key !== "sequence"){
                        return (
                            <React.Fragment  key={key}>
                                <label htmlFor={key}>{this.props.name[key]}</label>
                                <input name={key} value={this.state.params.key}
type={"text"} onChange={this.onChange} placeholder={this.props.placeholder[key]}/>
                                <br/>}})}
                <br/>
                <button className={"test-button"} onClick={this.onClick}>Тест</button>
            </div>
        );}}

export default InputForm;
```

Компонент сторінки комплексного тесту багатовимірних статистик AllMd.js

```
class AllMd extends Component{
    constructor(props) {
        super(props);
        this.state = {
            result: 0,
            index: 0,
            keys: []
        }
    }
    next = () =>{

        if (this.state.index < Object.keys(this.state.result).length-1){
```

```
            this.setState({index: ++this.state.index})
        }
    }
    prev = () =>{
        if (this.state.index > 0){
            this.setState({index: --this.state.index})
        }
    }
    update = (data)=>{
        this.setState({result:data,
                           keys: Object.keys(data).map((key =>(
                               key
                           ))),
                           index: 0})}
    render() {
        return (
            <div>
                <h2>Комплексний тест послідовності з використанням тестів заснованих на
багатовимірних статистиках</h2>

                <AllMdForm update ={this.update}/>
                <div id={"md-all-result"}>
                    <div>
                        {!Object.keys(this.state.result).length === false ?
                            <table>
                                {Object.keys(this.state.result).map(key => (
                                    <tr>
                                        <td style={{fontWeight:"bolder"}}>{data[key]}</td>
                                        <td>
                                            {this.state.result[key].value.toFixed(10)}
                                        </td>
                                    </tr>

                                ))}
                            </table> :
                            <div style={{padding: "20px"}}>Виконайте тести щоб отримати
результати</div>}
                    </div>

                    {this.state.result!==0 ? <div>
                        <PieChart name = {data[this.state.keys[this.state.index]]}

pValue={this.state.result[this.state.keys[this.state.index]]["value"]}/>
                            <div style={{textAlign: "center"}}>
                                <button className={"direction-button"}
                                        onClick={this.prev}>
                                    &#60;</button>
                                <button className={"direction-button"}
                                        onClick={this.next}>
                                    &#62;</button>
                            </div>
                    </div> : <div/>}
                </div>
            </div>        );}}
export default AllMd;
```

Додаток Е

Код моделі генератора випадкових чисел

```
#include <Wire.h>
int ADXL345 = 0x53;
const int SOUND_PIN = 8;
const int SAMPLE_TIME = 50;
unsigned long millisCurrentSound;
unsigned long millisLastSound = 0;
unsigned long millisElapsedSound = 0;
int sampleBufferValue = 0;
unsigned long millisCurrentTrigger;
unsigned long millisLastTrigger = 0;
unsigned long millisElapsedTrigger = 0;
int trigger = 1;
int sound[100];
int acX[100];
int acY[100];
int acZ[100];

void setup() {
  Serial.begin(9600);
  Wire.begin();
  Wire.beginTransmission(ADXL345);
  Wire.write(0x2D);
  Wire.write(8);
  Wire.endTransmission();
  delay(10);
}

void loop() {
  readSound(); readAcceleration(); generateRandomNumber();
}

void readSound() {
  millisCurrentSound = millis();
  millisElapsedSound = millisCurrentSound - millisLastSound;
  if (digitalRead(SOUND_PIN) == HIGH) {
    sampleBufferValue++;}
  if (millisElapsedSound > SAMPLE_TIME) {
    sampleBufferValue = map(sampleBufferValue, 0, 10, 0, 500);
    recordValue(sound, sampleBufferValue);
    sampleBufferValue = 0;
    millisLastSound = millisCurrentSound;}}

void readAcceleration() {
  Wire.beginTransmission(ADXL345);
  Wire.write(0x32);
  Wire.endTransmission(false);
  Wire.requestFrom(ADXL345, 6, true);
  int xOut = (int)(Wire.read() | Wire.read() << 8);
  int yOut = (int)(Wire.read() | Wire.read() << 8);
  int zOut = (int)(Wire.read() | Wire.read() << 8);
  recordValue(acX, xOut);
  recordValue(acY, yOut);
  recordValue(acZ, zOut);}

void generateRandomNumber() {
  millisCurrentTrigger = millis();
  millisElapsedTrigger = millisCurrentTrigger - millisLastTrigger;
```

```
  if (analogRead(A0) < 1000 && millisElapsedTrigger > 1000) {
    unsigned long total = 0;
    for (int i = 0; i < 99; i++) {
      total += sound[i];
      total = total << i % 2;
      total += acX[i];
      total = total << i % 3;
      total += acY[i];
      total = total << i % 4;
      total += acZ[i];
      total = total << i % 5;}
    Serial.println(toBinary(total));
    millisLastTrigger = millisCurrentTrigger;
  }}

String toBinary(long n) {
  String r;
  while (n != 0) {
    r = (n % 2 == 0 ? "0" : "1") + r;
    n /= 2;}
  return r;}

void recordValue(int a[], int value) {
  for (int i = 0; i < 99; i++) {
    a[i] = a[i + 1];}
  a[99] = value;}
```

Код моделі генератора випадкових чисел з використанням Fisher-Yates shuffle

```
void generateRandomNumber(){
  millisCurrentTrigger = millis();
  millisElapsedTrigger = millisCurrentTrigger - millisLastTrigger;
    if(analogRead(A0) < 1000 && millisElapsedTrigger > 1000){
        shuffle(sound);
        shuffle(acX);
        shuffle(acY);
        shuffle(acZ);
        unsigned long total = 0;
        for(int i = 0; i < 99; i++){
            total += sound[i];
            total = total << 1;
            total += acX[i];
            total = total << 1;
            total += acY[i];
            total = total << 1;
            total += acZ[i];
            total = total << 1;}
        Serial.println(toBinary(total));
        millisLastTrigger = millisCurrentTrigger;
      }}

void shuffle(int arr[]){
    for(int i = 99; i > 0; i--){
       int index = (int) random(i+1);
       int temp = arr[i];
       arr[i] = arr[index];
       arr[index] = temp;
    }}
```

Додаток Ж

Експериментальні дані та результати тестування послідовностей створених генераторами

Таблиця Ж.1

Дані та результати для ГПЧ

Випадкова послідовність	Результат тесту							
	1	2	3	4	5	6	7	8
11101010110101000011000010101000	0	0	0	0	0	0	0	0
11101111001110010010000111111000	0	0	0	0	0	0	0	0
10100100011010111101100000001000	0	1	0	0	0	0	0	0
10100100001100110100001011110100	1	1	1	0	1	0	0	1
11000100010110100110001001100000	1	0	0	0	0	0	0	0
11110110000011110000111010101100	0	0	0	0	0	0	0	1
11110000110010110010001011100100	1	0	0	1	1	0	0	0
10101000010111110001101110100100	0	1	0	1	1	0	0	0
11000100010011110000111001100000	0	0	0	0	0	0	0	0
11010101110000001011100010111100	0	0	0	0	0	0	0	0
Всього:	3	3	1	2	3	0	0	2

Таблиця Ж.2

Дані та результати для АГВЧ

Випадкова послідовність	Результат тесту							
	1	2	3	4	5	6	7	8
11001101000111010011100001011000	1	1	1	1	1	1	1	1
11101111011001011000101001000100	1	1	1	1	1	0	0	1
11100110000110000000100000011000	0	0	0	0	0	0	0	0
10000001110000000111011111010010	0	0	0	0	0	0	0	0
11010010000111101011110101001100	0	1	0	1	1	0	0	0

Продовження табл. Ж.2

11101000001010001101011101000 10	0	1	0	0	0	0	0	0
10001000101010011100101110101 10	0	0	0	0	0	0	0	0
11001010110110011010101100100 10	0	0	0	0	0	0	0	0
10110010101110101000100101110 10	0	0	0	0	0	0	0	0
10000011010010101101000110001 10	1	1	1	0	0	1	0	0
Всього:	3	5	3	3	3	2	1	2

Таблиця Ж.3

Результати тестування АГВЧ, що використовує алгоритм Fisher-Yates shuffle

Випадкова послідовність	Результат тесту							
	1	2	3	4	5	6	7	8
10011010110100000011111000010 10	0	1	0	0	0	0	0	0
10101001001110001100000110010 10	1	0	0	0	1	0	0	1
11100011001110001000011011010 10	1	1	1	1	1	1	1	1
10010011010111100101000111000 10	1	0	0	1	1	0	0	1
10100101001100101010010101101 10	0	0	0	0	0	0	0	0
11101100011001011100110000110 00	0	0	0	0	0	0	0	0
11011000101011011010000100111 10	1	1	1	1	1	1	1	0
11110001001100110100011100110 10	0	0	0	1	0	1	0	0
10011010101111011001111000111 10	1	1	0	0	0	0	0	0
10100001011001011111011001100 10	1	1	1	1	1	1	1	1
Всього:	6	5	3	5	5	4	3	4

Таблиця Ж.4

Результати тестування АГВЧ в стані відносного спокою

Випадкова послідовність	Результат тесту							
	1	2	3	4	5	6	7	8
10011010111000001110001001000 00	0	0	0	0	0	0	0	0

Продовження табл. Ж.4

11110111011101100111110010011000	0	0	0	0	0	0	0	0
10010101100010001001100101010000	0	0	0	0	0	0	0	0
11000101101001110011001100110110	0	1	0	0	0	0	0	0
10111010001100110110111101111110	1	0	0	0	0	0	0	0
10101101000010000010111111101000	0	0	0	0	0	0	0	0
10100110000010011101111101100010	1	1	1	1	1	1	1	1
11100011001001100110111111110100	0	0	0	0	0	0	0	0
11011000100111001000001100010000	0	0	0	0	0	0	0	0
11011110101001110111010001111110	0	0	0	0	0	1	0	0
Всього:	2	2	1	1	1	2	1	1