Галицький фаховий коледж імені В’ячеслава Чорновола

ЗБІРНИК ТЕЗ

за матеріалами студентської науково-практичної конференції в рамках Днів Науки 2025 в Галицькому фаховому коледжі імені В’ячеслава Чорновола

ДНІ НАУКИ 2025

секція «Комп'ютерних технологій»

Тернопіль, 2025

ЗБІРНИК тез за матеріалами студентської науково-практичної конференції в рамках Днів Науки 2025 в Галицькому фаховому коледжі імені В'ячеслава Чорновола (секція «Комп'ютерних технологій» – Тернопіль: Галицький фаховий коледж імені В'ячеслава Чорновола, 2025. 112 с.

До збірника увійшли матеріали і тези доповідей, які подано учасниками студентських науково-практичних конференцій в рамках Днів Науки в Галицькому фаховому коледжі імені В'ячеслава Чорновола (секція «Комп'ютерних технологій»).

При публікації максимально збережено орфографію, пунктуацію та стилістику запропоновану авторами та підтриману науковими керівниками. За зміст праць та достовірність наведених фактів і статистичних даних відповідальність несуть автори та їх наукові керівники.

Рекомендовано до друку Науково-методичною радою з питань якості освіти Галицького фахового коледжу імені В'ячеслава Чорновола

Верстка: Черних Д.

ЗМІСТ

УДК 004.89

***Береза Олександр**, здобувач вищої освіти*
IV курсу спеціальності 123 «Комп'ютерна інженерія»,
керівник роботи спеціаліст вищої категорії,
викладач-методист, Павлюс В.П.

ІНТЕЛЕКТУАЛЬНА СИСТЕМА ГОЛОСОВОГО КЕРУВАННЯ ПРИСТРОЯМИ "РОЗУМНОГО" ДОМУ

Що таке "розумний дім" та "розумна екосистема"? Поняття "розумного дому" (smart home) описує середовище, в якому побутові пристрої інтегровані в єдину мережу для автоматичного або дистанційного керування. Це може включати освітлення, опалення, побутову техніку, камери спостереження, штори – все це під контролем користувача через застосунок, голос або навіть за заданими правилами.

Розумна екосистема – це вже вищий рівень інтеграції, де декілька систем (освітлення, безпека, енергозбереження, голосові помічники тощо) взаємодіють між собою, адаптуючись до поведінки людини, її звичок і потреб.

Сучасні рішення в галузі розумного дому можна умовно поділити на кілька типів:

1. Комерційні платформи: Такі як Google Home, Amazon Alexa, Apple HomeKit – зручні для масового користувача, мають зручний інтерфейс, але обмежені у кастомізації, часто потребують платних підписок та мають закриту архітектуру.

2. Open-source рішення: Наприклад, Home Assistant, OpenHAB – орієнтовані на ентузіастів і розробників, дозволяють повну свободу конфігурації, працюють локально, але вимагають технічних знань.

3. DIY-системи на базі мікроконтролерів: Проєкти на ESP32, Arduino, Raspberry Pi – дозволяють створити повністю індивідуалізовану систему, з урахуванням усіх побажань користувача. Часто комбінуються з відкритими протоколами, як-от MQTT, і хмарними API.

Першими, хто задав тренд у розумному будинку, були Google та Amazon. Їх системи Google Home і Alexa вивели автоматизацію на новий рівень:

– Google зробив ставку на Android-екосистему та хмарну обробку голосу.

– Amazon відкрив можливості для сторонніх розробників через систему “Alexa Skills”.

– Паралельно з цим з'явився Home Assistant – open-source альтернатива, яка дозволила реалізовувати подібні функції локально та без залежності від корпорацій.

Однак у всіх згаданих систем є мінуси: обмежена кастомізація, висока вартість, необхідність постійного підключення до хмари, а іноді ще й потрібно прикласти багато зусиль для налаштування.

Враховуючи зазначені недоліки було прийнято рішення розробити альтернативну систему з наступними характеристиками:

– доступна для повторення;

– гнучка у налаштуванні;

– працюватиме на відкритих протоколах;

– включатиме голосове керування з інтеграцією ШІ.

Результатом розробки стала система, реалізована на базі мікроконтролера ESP32 та з такими особливостями:

– Допоміжні модулі реалізовані на базі ESP8266-01 із датчиками (DHT11 – для зчитування температури та вологості, MQ-135 – для зчитування загальної якості повітря в приміщенні, модуль датчика освітлення (фоторезистора) – для зчитування рівня освітлення на вулиці,

– Використано протокол MQTT для обміну повідомленнями між пристроями та застосунком;

– Firebase Firestore використано в ролі СУБД;

– Google Speech-to-Text / Text-to-Speech використано для розпізнавання тексту та озвучення;

– ChatGPT API використано для інтерпретації запитів.

Мобільний застосунок для Android, що розроблений на базі Dart/Flutter, є головною панеллю керування. Через нього користувач може:

– керувати освітленням (вручну або через пресети);

– переглядати дані з датчиків та інформацію про навколишнє середовище;

– задавати голосові або текстові команди (з їх обробкою за допомогою інтеграції ШІ);

– створювати автоматизацію за часом або умовами, наприклад: “якщо температура > 25°C – вмикай синє світло”, або “кожного дня о 10:00 вмикай лагідне світло”.

Система підтримує розширення та адаптацію під конкретного користувача. Вона також орієнтована на людей з обмеженими можливостями, адже дозволяє повноцінне керування за допомогою лише голосу.

На рисунках 1 та 2 схематично представлено структуру та основні процеси системи.

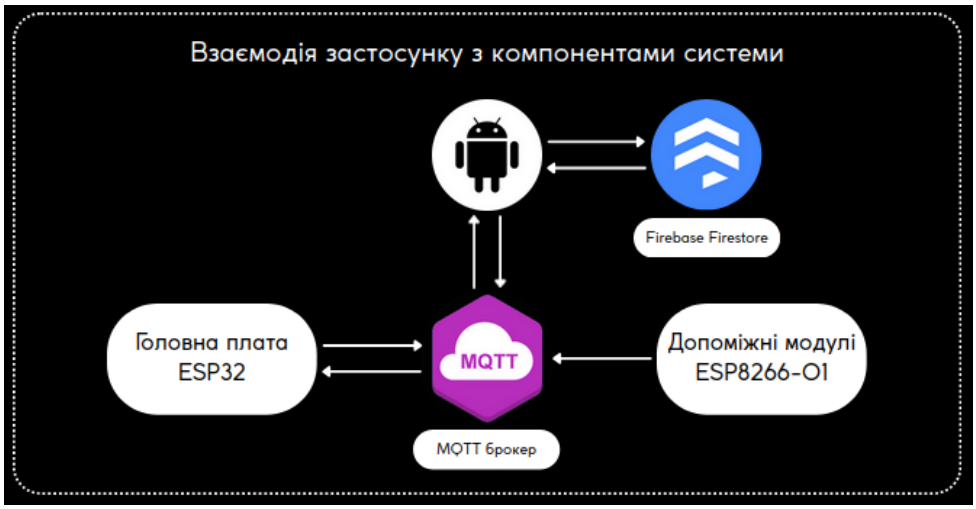

Рисунок 1 – Схема взаємодії застосунку та компонентів системи

Рисунок 2 – Схема обробки голосового запиту

Розроблена система доводить, що сьогодні майже кожен ентузіаст може створити власний розумний дім. Вона об’єднує найкращі риси комерційних та open-source рішень: гнучкість, автономність, мінімальні витрати та широкі можливості кастомізації.

СПИСОК ДЖЕРЕЛ ПОСИЛАННЯ

1. Розумний дім. Wikipedia. Вебсайт. URL: https://uk.wikipedia.org/wiki/Розумний_дім (дата звернення:. 05.05.2025).

2. Google Home. Wikipedia. Вебсайт. URL: https://uk.wikipedia.org/wiki/Google_Home (дата звернення:. 09.05.2025).

3. Amazon Alexa. Wikipedia. Вебсайт. URL: https://uk.wikipedia.org/wiki/Amazon_Alexa (дата звернення:. 10.05.2025).

4. Home Assistant. Wikipedia. Вебсайт. URL: https://uk.wikipedia.org/wiki/Home_Assistant (дата звернення:. 15.05.2025).

УДК 004

***Білик Ярослав**, здобувач ОКР фаховий молодший бакалавр*

IV курсу спеціальності «Комп'ютерні науки»

науковий керівник Гавришків Н.Г.

ДОСЛІДЖЕННЯ ОПТИМІЗАЦІЇ ШЕЙДЕРІВ ДЛЯ ГРАФІЧНИХ ЗАСТОСУНКІВ

Сучасний прогрес у сфері графічних технологій стимулює підвищення стандартів продуктивності для графічних застосунків [1]. Збільшення складності візуальних ефектів, що щодня обробляються мільйонами застосунків, робить оптимізацію шейдерів критично важливим завданням [2]. Ця робота присвячена аналізу основних проблем, пов'язаних із продуктивністю шейдерів у графічних застосунках, дослідженню методів їх оптимізації, таких як мінімізація обчислень, ефективні алгоритми, управління пам'яттю та адаптація до апаратного забезпечення, а також розробці рекомендацій для підвищення продуктивності [3]. Об'єктом дослідження є графічні застосунки, що використовують шейдери для рендерингу, а предметом — технології та методи оптимізації шейдерів для забезпечення високої ефективності [4]. Використані методи включають теоретичний аналіз,

індукцію, дедукцію та аналіз практичних кейсів [5]. Наукова новизна полягає в систематизації сучасних методів оптимізації шейдерів, оцінці їхньої ефективності та адаптації до нових графічних технологій [6].

Графічні застосунки, що використовують шейдери, стикаються з низкою викликів, які впливають на продуктивність та якість візуалізації [7]. Основні проблеми включають:

– Висока обчислювальна складність: Складні шейдери з великою кількістю операцій створюють значне навантаження на графічний процесор (GPU), особливо на мобільних пристроях із обмеженими ресурсами. Наприклад, у грі "Genshin Impact" розробники оптимізували шейдери для стабільної частоти кадрів на смартфонах середнього класу [4].

– Неефективне використання пам’яті: Шейдери, що працюють із великими текстурами, перевантажують пам’ять GPU, знижуючи продуктивність. У 2021 році в Unreal Engine оптимізували доступ до пам’яті для зменшення затримок [8].

– Погана оптимізація коду: Зайві цикли чи умовні оператори в коді шейдерів уповільнюють виконання. Дослідження NVIDIA показали прискорення рендерингу до 25% при видаленні непотрібних обчислень [5].

– Недостатня адаптація до апаратного забезпечення: Шейдери, не враховуючи специфіку GPU, не використовують його потенціал повною мірою. У "DOOM Eternal" адаптація шейдерів під різні платформи оптимізувала гру для широкого спектра пристроїв [6].

– Проблеми з паралелізмом: Неоптимізовані для багатопоточності шейдери обмежують можливості GPU. Дослідження Khronos Group у 2020 році показало зростання продуктивності на 30% при паралелізації [3].

– Високе споживання енергії: Складні шейдери на мобільних пристроях швидко розряджають батарею. Apple у 2022 році оптимізувала шейдери в Metal API для зменшення енергоспоживання на iPhone [1].

Для узагальнення даних типові проблеми продуктивності шейдерів представлено в таблиці 1.

Таблиця 1 – Проблеми продуктивності шейдерів

Проблема	Опис	Приклад
Висока обчислювальна складність	Перевантаження GPU	"Genshin Impact"
Неефективне використання пам’яті	Великі текстури	Unreal Engine
Погана оптимізація коду	Зайві операції	NVIDIA дослідження
Недостатня адаптація до апаратного забезпечення	Невикористання повного потенціалу GPU	Адаптація шейдерів у "DOOM Eternal"
Проблеми з паралелізмом	Неоптимізовані для багатопоточності шейдери	Дослідження Khronos Group 2020
Високе споживання енергії	Швидкий розряд батареї на мобільних пристроях	Оптимізація в Metal API для iPhone

Історія Розвитку Шейдерів

Розвиток шейдерів розпочався з появою перших графічних конвеєрів у 1980-х роках, коли графічні процесори (GPU) використовували фіксовані

функції для обробки графіки. Ці ранні системи, такі як OpenGL 1.0 (1992) і DirectX 2.0 (1996), обмежували розробників у створенні складних ефектів, оскільки параметри рендерингу задавалися апаратно [9]. Переломним моментом став 2001 рік, коли Microsoft представила DirectX 8.0, що підтримувала програмовані шейдери з мовою HLSL (High-Level Shader Language), а OpenGL 1.5 (2003) додав підтримку GLSL (OpenGL Shading Language) [9]. Це дозволило розробникам створювати власні алгоритми для обробки вершин і пікселів, відкриваючи двері до гнучкішого рендерингу.

У 2000-х роках шейдери стали невід'ємною частиною ігрової індустрії. Ігри, такі як Doom 3 (2004), активно використовували шейдери для реалістичного освітлення та текстур [6]. З появою DirectX 10 (2006) і OpenGL 3.0 (2008) з'явилися геометричні шейдери, які розширили можливості обробки геометрії [3]. У 2010-х роках розвиток пішов у бік обчислювальних шейдерів, що дозволили використовувати GPU для паралельних обчислень поза рендерингом, наприклад, у симуляціях чи машинному навчанні [3].

Сучасний етап розвитку шейдерів пов'язаний із появою нових API, таких як Vulkan (2015) і Metal (2014), які забезпечують низькорівневий контроль над GPU, що сприяє оптимізації [7]. Інтеграція з технологіями ray tracing, як у DirectX Raytracing (2018), додала шейдерам нові можливості для трасування променів у реальному часі [6]. Сьогодні шейдери еволюціонують разом із потребами VR, AR та мобільних ігор, де важлива не лише якість графіки, а й енергоефективність [1]. Ця еволюція підкреслює важливість оптимізації шейдерів у сучасних графічних застосунках.

Типи Шейдерів Та Їх Функції

Шейдери є ключовими компонентами сучасного графічного конвеєра, виконуючи різні етапи обробки даних. Існує кілька основних типів шейдерів, кожен із яких має специфічну роль у рендерингу [9]. Розуміння їхнього

функціоналу є важливим для оптимізації продуктивності графічних застосунків.

– Вершинні шейдери (Vertex Shaders): Ці шейдери обробляють 3D-координати вершин об’єктів, виконуючи трансформації (наприклад, переміщення, обертання, масштабування) у просторі камери. Вони є першим етапом у конвеєрі та визначають геометрію сцени [3]. Наприклад, у грі Doom Eternal вершинні шейдери оптимізуються для динамічної обробки складних моделей [6].

– Фрагментні шейдери (Fragment Shaders): Відповідають за розрахунок кольору та текстури для кожного пікселя, який утворюється після растеризації. Вони додають ефекти освітлення, тіні та постобробки, що робить їх критичними для візуальної якості [9]. Оптимізація фрагментних шейдерів, як у Genshin Impact, дозволяє досягти стабільної частоти кадрів на мобільних пристроях [4].

– Геометричні шейдери (Geometry Shaders): Ці шейдери працюють на рівні примітивів (трикутників, ліній), дозволяючи створювати чи модифікувати геометрію. Вони корисні для генерації частинок чи тесселяції, але їхнє використання може підвищувати навантаження на GPU [3]. Прикладом є їхнє застосування у DirectX 10 для створення ефектів тіні [6].

– Обчислювальні шейдери (Compute Shaders): На відміну від інших типів, обчислювальні шейдери не прив’язані до рендерингу й використовуються для паралельних обчислень, таких як симуляція фізики, обробка даних чи машинне навчання. Вони доступні через API, такі як Vulkan і OpenGL, і є потужним інструментом для оптимізації [3].

Для ілюстрації процесу обробки даних у графічному конвеєрі наведено діаграму послідовності (рис. 1), яка відображає перехід даних через шейдери.

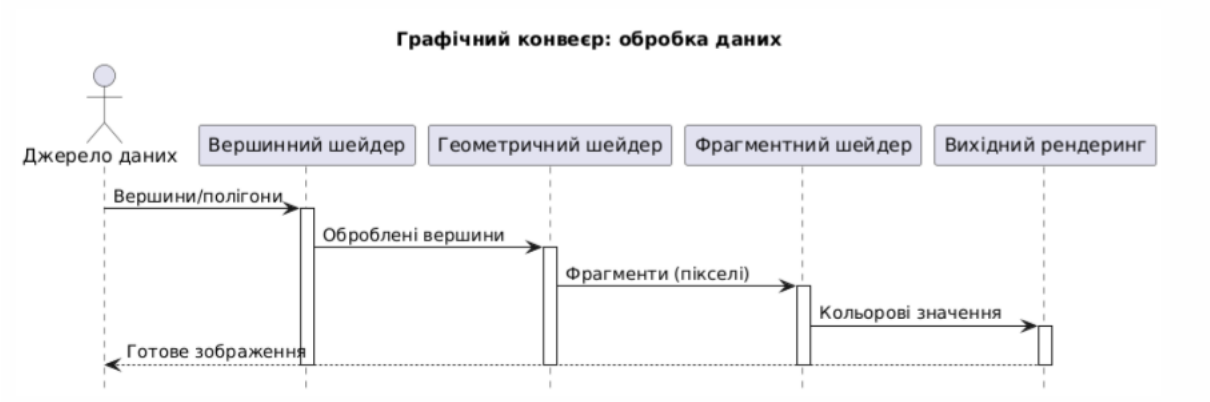

Рисунок 1 – Діаграма послідовності обробки даних у графічному конвеєрі

Крім того, діаграма потоків (рис. 2) демонструє етапи оптимізації шейдерів, що допомагає зрозуміти практичний підхід до підвищення їхньої ефективності

Рисунок 2 – Діаграма послідовності обробки даних у графічному конвеєрі

Проведене дослідження підкреслило, що оптимізація шейдерів є одним із ключових факторів, які визначають продуктивність і ефективність сучасних графічних застосунків, особливо на пристроях із обмеженими

обчислювальними ресурсами, таких як мобільні гаджети [1]. Аналіз основних проблем, таких як висока обчислювальна складність, неефективне використання пам’яті та недостатня адаптація до апаратного забезпечення, показав, що ці аспекти суттєво впливають на швидкість рендерингу та споживання енергії [4]. Використання методів оптимізації, таких як мінімізація обчислень, спрощення алгоритмів і вдосконалення управління пам’яттю, дозволяє значно підвищити продуктивність шейдерів, що підтверджується теоретичними висновками та аналізом практичних кейсів [5].

Робота продемонструвала, що систематизація сучасних підходів до оптимізації шейдерів та їх адаптація до нових графічних технологій має значний науковий і практичний потенціал. Розробники можуть використовувати отримані результати для створення графічних застосунків із кращою швидкодією та зниженим енергоспоживанням, що особливо актуально для мобільних платформ і віртуальної реальності [6]. Особливу увагу приділено необхідності врахування специфіки апаратного забезпечення, що сприяє максимальному розкриттю потенціалу GPU та зменшенню навантаження на систему [6].Попри досягнуті результати, залишаються виклики, пов’язані з паралелізмом обчислень і високим енергоспоживанням складних шейдерів, що потребує подальших досліджень [3]. Отримані напрацювання можуть стати основою для розробки інноваційних інструментів і методик, що сприятимуть подальшому розвитку галузі комп’ютерної графіки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Advanced metal shader optimization - WWDC16 - videos - apple developer. Apple Developer. URL:https://developer.apple.com/videos/play/wwdc2016/606/ (дата звернення: 09.03.2025).

2. A system for rapid exploration of shader optimization choices | research. Research at NVIDIA | Advancing the Latest Technology | NVIDIA. URL: https://research.nvidia.com/publication/2016-07_system-rapid-exploration-shader-optimization-choices (дата звернення: 09.03.2025).

3. Compute shader - opengl wiki. The Khronos Group Inc. URL: https://www.khronos.org/opengl/wiki/Compute_Shader (дата звернення: 09.03.2025).

4. How genshin impact 3D models are optimized for mobile performance. Animatics Asset Store. URL: https://www.animaticsassetstore.com/2024/09/13/how-genshin-impact-3d-models-are-optimized-for-mobile-performance/ (дата звернення: 09.03.2025).

5. Improve shader performance and in-game frame rates with shader execution reordering | NVIDIA technical blog. NVIDIA Technical Blog. URL: https://developer.nvidia.com/blog/improve-shader-performance-and-in-game-frame-rates-with-shader-execution-reordering/ (дата звернення: 17.06.2025).

6. Linneman J., Judd W. Creating Doom: The Dark Ages - ray tracing, load times and optimisation in id Tech 8. Eurogamer.net. URL: https://www.eurogamer.net/digitalfoundry-2025-creating-doom-the-dark-ages-how-id-tech-8-took-shape (дата звернення: 09.03.2025).

7. Metal performance shaders | apple developer documentation. Apple Developer Documentation. URL: https://developer.apple.com/documentation/metalperformanceshaders (дата звернення: 09.03.2025).

8. Optimizing shaders in unreal engine. Luna's Technical Art Blog. URL: https://calvinatorrtech.art.blog/2023/12/20/optimizing-shaders-in-unreal-engine/ (дата звернення: 09.03.2025).

9. Shader - OpenGL Wiki. The Khronos Group Inc. URL: https://www.khronos.org/opengl/wiki/shader (дата звернення: 09.03.2025).

10. Unity - manual: optimize shaders. Unity - Manual: Unity 6.1 User Manual. URL: https://docs.unity3d.com/Manual/SL-ShaderPerformance.html (дата звернення: 09.03.2025).

УДК 004.75

***Бончук Олександр**, здобувач вищої освіти*
IV курсу спеціальності 123 «Комп'ютерна інженерія»,
керівник роботи спеціаліст вищої категорії,
викладач-методист, Павлюс В.П.

РОЗПОДІЛЕНА СИСТЕМА МОНІТОРИНГУ ПОКАЗНИКІВ ПОВІТРЯ В ЗАКРИТОМУ ПРИМІЩЕННІ

Система моніторингу показників повітря складається з головного та сенсорних модулів із власними джерелами живлення. Головний модуль містить мікроконтролер ESP32 та кольоровий TFT-дисплей ILI9342 (SPI) для візуалізації даних, а також бузер для звукового сигналу. ESP32 – це потужний контролер з двома ядрами Tensilica LX6 (до 240 МГц) та вбудованим Wi-Fi/Bluetooth. Він збирає й обробляє дані від усіх сенсорних вузлів і відображає результати на екрані в реальному часі. Сенсорні модулі базуються на мікроконтролері ESP8266-01 –дешевому Wi-Fi SOC з вбудованим стеком TCP/IP, що дозволяє їм автономно вимірювати навколишні параметри (температура, вологість, тиск, забруднення повітря) та передавати результати на центральний модуль по бездротовій мережі. Така розподілена модульна архітектура забезпечує

гнучкість і масштабованість системи: за потреби можна додавати нові сенсорні вузли, не змінюючи основну логіку.

Програмна логіка. Система працює зациклено: сенсорні модулі зчитують параметри довкілля і передають їх на головний контролер, де дані аналізуються і виводяться користувачеві. Зчитування та обробка відбуваються за таким алгоритмом:

− кожен сенсорний модуль (ESP8266) періодично вимірює температуру, відносну вологість, атмосферний тиск і рівні забруднення PM2.5/PM10, а потім надсилає отримані показники на головний модуль по Wi-Fi;

− головний модуль (ESP32) приймає ці дані, порівнює їх із заздалегідь заданими нормами та показує значення на TFT-дисплеї;

− індикація стану здійснюється кольоровим світлодіодом: зелений – все в нормі, жовтий – попереджувальні відхилення, червоний – критичні показники, що вимагають уваги. У разі критичного перевищення будь-яких параметрів одночасно активується бузер для звукового оповіщення (аварійний сигнал).

Таким чином, система забезпечує безперервний моніторинг і своєчасне сповіщення про будь-які відхилення від заданих меж.

Підбір компонентів. Для реалізації системи вибрано такі компоненти:

− DHT22 (AM2302) – цифровий датчик температури та вологості. Діапазон вимірюваних температур – від -40°C до +80°C з точністю ±0.5°C, а вологості – 0-100% RH з точністю ~2-5%. Це дозволяє досить точно відстежувати умови в приміщенні.

− BMP-180 – високоточний барометричний сенсор тиску (Bosch). Він вимірює тиск в межах 300–1100 гПа з роздільною здатністю до 0.03 гПа

(±0.25 м) і температурою –40…+85°C (точність ±2°C). Така точність необхідна для виявлення незначних змін атмосферного тиску.

− PMS5003 (Plantower) – лазерний датчик концентрації часток з діапазоном вимірюваних PM1.0, PM2.5 і PM10. Він має вбудований вентилятор і лазер, що вимірює кількість та розмір частинок у повітрі. Цей сенсор дозволяє отримувати цифрові показники забруднення повітря (PM2.5/PM10) для оцінки якості повітря в приміщенні.

− ESP8266-01 – малопотужний Wi-Fi мікроконтролер, що забезпечує зв’язок сенсорного модуля з мережею. Він автономно працює від свого живлення і передає дані на центральний модуль через TCP/IP.

− ESP32 – центральний контролер головного модуля. Має два ядра Tensilica LX6 (до 240 МГц), вбудований Wi-Fi та Bluetooth. Використовується для отримання даних з усіх вузлів, їх обробки та виведення результатів.

− TFT-дисплей ILI9342 (2.2" SPI, 320×240) – кольоровий графічний дисплей для відображення поточних показників у реальному часі. Підключений по інтерфейсу SPI для швидкої передачі зображень.

− RGB-світлодіод – підключений до ESP32, забезпечує візуальне оповіщення про стан системи (зелений, жовтий, червоний).

− Бузер – звуковий сигналізатор, що при активації генерує звуковий сигнал при перевищенні критичних порогів.

Сценарії та поведінка системи. Сценарії описують реакцію системи на зміни навколишніх умов. Спочатку задаються допустимі межі для кожного параметра. Наприклад, оптимальна температура в приміщенні – 20-22°C, вологість – 30-50%. При температурі вище 24°C з’являється відчуття дискомфорту, а понад 30°C – значне зниження працездатності й ризик теплового удару. Вологість повітря нижча за 30% викликає сухість шкіри та слизових, тоді як вище 70% створює умови для росту цвілі та пилових кліщів. Атмосферний

тиск вважається нормальним на рівні близько 1013 гПа; відхилення більш ніж на ±10% (близько ±100 гПа) може впливати на самопочуття людей, спричиняючи головний біль у чутливих осіб. Щодо забруднення повітря, ВООЗ рекомендує рівні ПМ2.5 не більше 10 мкг/м³ і ПМ10 не більше 20 мкг/м³ (середньорічні). При досягненні цих меж система реагує відповідно: при нормальних значеннях – зелений індикатор, при попереджувальних перевищеннях – жовтий, при критичних – червоний плюс звуковий сигнал. Також запрограмовано реакції на різкі зміни: наприклад, при падінні тиску нижче граничного – система може акцентувати це на дисплеї чи звуковому попередженні.

Етапи розробки сценаріїв:

− встановлення нормативних порогів для кожного параметра (температура, вологість, тиск, ПМ), з огляду на гігієнічні рекомендації та стандарти;

− визначення дій системи при перетині кожного порогу: звуковий/візуальний сигнал, зміна кольору індикації, оновлення повідомлень на дисплеї;

− моделювання алгоритму роботи (блок-схема): послідовність перевірки живлення, зчитування даних, їх передача, аналіз порогів і сигналізація;

− тестування сценаріїв у реальних умовах для налаштування чутливості та своєчасності оповіщень.

Висновки. У результаті проведеного дослідження та реалізації проєкту розподіленої системи моніторингу повітря в закритому приміщенні можна зробити такі висновки:

− розподілена архітектура забезпечує масштабованість, надійність і гнучкість системи, що дозволяє легко адаптувати її до різних умов експлуатації та розширювати без кардинальних змін конструкції;

– використання сучасних мікроконтролерів ESP32 та ESP8266-01 у поєднанні з Wi-Fi-з’єднанням дозволяє організувати автономну, енергоефективну та стабільну передачу даних між модулями системи;

– інтеграція сенсорів DHT22, BMP180 та PMS5003 забезпечує точне й достовірне вимірювання температури, вологості, атмосферного тиску та рівня забруднення повітря у вигляді часток PM2.5 та PM10, що є критично важливим для контролю мікроклімату в побутових, офісних або виробничих приміщеннях;

– механізм візуального (RGB-індикація) і звукового (бузер) оповіщення дозволяє оперативно інформувати користувача про перевищення гранично допустимих значень параметрів повітря, забезпечуючи своєчасне реагування на потенційні загрози.

Таким чином, створена система є ефективним інструментом для підвищення екологічної безпеки та комфорту в приміщеннях і має перспективи подальшого вдосконалення, зокрема шляхом додавання нових сенсорів, інтеграції з мобільними додатками або хмарними сервісами.

СПИСОК ДЖЕРЕЛ ПОСИЛАННЯ:

1. Характеристики датчика температури і вологості DHT22. SparkFun Electronics: вебсайт. URL: https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf (дата звернення: 05.05.2025).

2. Характеристики барометричного сенсора BMP180. Adafruit : веб-сайт. URL: https://cdn-shop.adafruit.com/datasheets/BST-BMP180-DS000-09.pdf (дата звернення: 05.05.2025).

3. Інформація про лазерний датчик якості повітря PMS5003. AQICN.org : веб-сайт. URL: https://aqicn.org/sensor/pms5003-plantower/ (дата звернення: 05.05.2025).

4. Технічна документація ESP8266EX. Espressif Systems : веб-сайт. URL: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf (дата звернення: 05.05.2025).

5. Технічний довідник по ESP32. Espressif Systems : веб-сайт. URL: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf (дата звернення: 05.05.2025).

6. Норма температурного режиму та вмісту хімічних і біологічних речовин в атмосферному повітрі населених місць. НАКАЗ МОЗ України від 10.05.2024 № 813 "Про затвердження державних медико-санітарних нормативів допустимого вмісту хімічних і біологічних речовин в атмосферному повітрі населених місць": веб-сайт. URL: https://zakon.rada.gov.ua/laws/show/z0763-24#n9 (дата звернення 05.05.2025).

7. Що таке PM2.5 і PM10 та як впливають на здоров'я. ВООЗ – Всесвітня організація охорони здоров'я : веб-сайт. URL: https://www.who.int/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health (дата звернення: 05.05.2025).

УДК 004

***Велещук Андрій**, здобувач ОКР фаховий молодший бакалавр*
IV курсу спеціальності «Комп'ютерні науки»
науковий керівник Гавришків Н.Г.

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ РІЗНИХ МЕТОДІВ СИНХРОНІЗАЦІЇ ДАНИХ МІЖ МОБІЛЬНИМ ТА ВЕБ-ДОДАТКАМИ ДЛЯ УПРАВЛІННЯ ЗАВДАННЯМИ.

Сучасні мобільні та веб-застосунки для управління завданнями потребують надійних механізмів синхронізації даних між різними пристроями та платформами. Актуальність цієї теми зумовлена стрімким розвитком мобільних технологій та зростанням очікувань користувачів щодо безперебійного доступу до даних у будь-який час та з будь-якого пристрою. Синхронізація виконує роль "невидимого мосту", який забезпечує цілісність інформації, узгодженість змін та послідовність взаємодії користувача з системою. Будь-які порушення в цьому процесі можуть призводити до дублювання завдань, втрати даних або навіть повної недієздатності системи.

Основні вимоги до синхронізації в to-do застосунках включають: підтримку офлайн-режиму, мінімальні затримки, енергоефективність та масштабованість. Для їх задоволення розробники використовують різні архітектурні підходи. Найпоширенішою є клієнт-серверна модель, де дані централізовано зберігаються на сервері, а клієнти (мобільний чи веб-застосунок) звертаються до нього за оновленнями. Ця модель добре масштабується, але вимагає постійного з'єднання з інтернетом. Для роботи в офлайн-режимі часто використовують гібридні архітектури з локальними базами даних (SQLite, Realm), які періодично синхронізуються з сервером.

Існує кілька основних методів синхронізації даних. Polling (періодичні запити) - простий у реалізації, але створює велике навантаження на сервер. Long polling покращує ефективність, утримуючи з'єднання до появи змін. WebSocket забезпечує двосторонній зв'язок у реальному часі, що ідеально підходить для миттєвих оновлень, але вимагає більше ресурсів. Push-повідомлення дозволяють сповіщати клієнтів про зміни без постійних запитів. Для складних сценаріїв з офлайн-редагуванням використовують спеціальні алгоритми вирішення конфліктів, такі як операційні трансформації або CRDT (Conflict-free Replicated Data Types).

Однією з ключових проблем синхронізації є вирішення конфліктів, коли одні й ті самі дані змінюються на різних пристроях одночасно. Для цього застосовують різні стратегії: "остання зміна перемагає", версіювання даних або ручне об'єднання. Інші виклики включають затримки синхронізації, втрату даних при нестабільному з'єднанні та оптимізацію енергоспоживання на мобільних пристроях.

Для візуалізації архітектури синхронізації було використано діаграму компонентів (рисунок 1), яка відображає взаємодію між клієнтом, сервером і локальним сховищем.

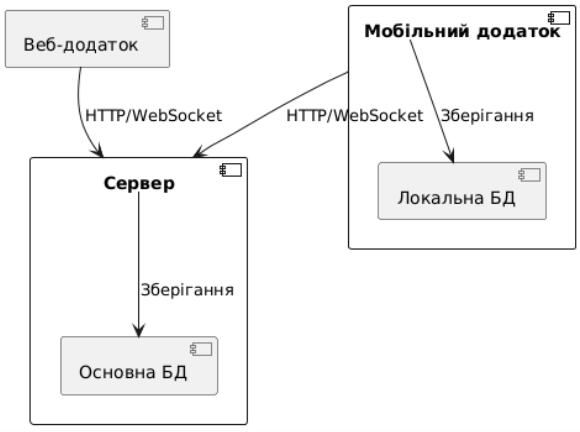

Рисунок 1. Діаграма компонентів

У типовому сценарії взаємодії користувача з to-do системою мобільний застосунок виконує роль ініціатора змін, які передаються на сервер. Сервер, у свою чергу, виконує роль централізованого посередника, що обробляє запити, зберігає дані та забезпечує їх доступність для інших клієнтів, зокрема веб-додатку. Якщо в системі реалізовано механізм push-повідомлень або WebSocket, сервер здатен майже миттєво повідомляти інші клієнти про внесені зміни, що значно підвищує загальну узгодженість даних у системі.

На діаграмі послідовності (рис. 2) зображено послідовність дій при додаванні нового завдання користувачем у мобільному додатку. Після внесення змін клієнтська частина надсилає їх на сервер, який, обробивши запит, надсилає відповідні push- або інші сповіщення до веб-клієнта. Веб-застосунок при отриманні такого сигналу виконує запит до сервера для отримання актуальних даних, що дозволяє підтримувати узгодженість між усіма платформами.

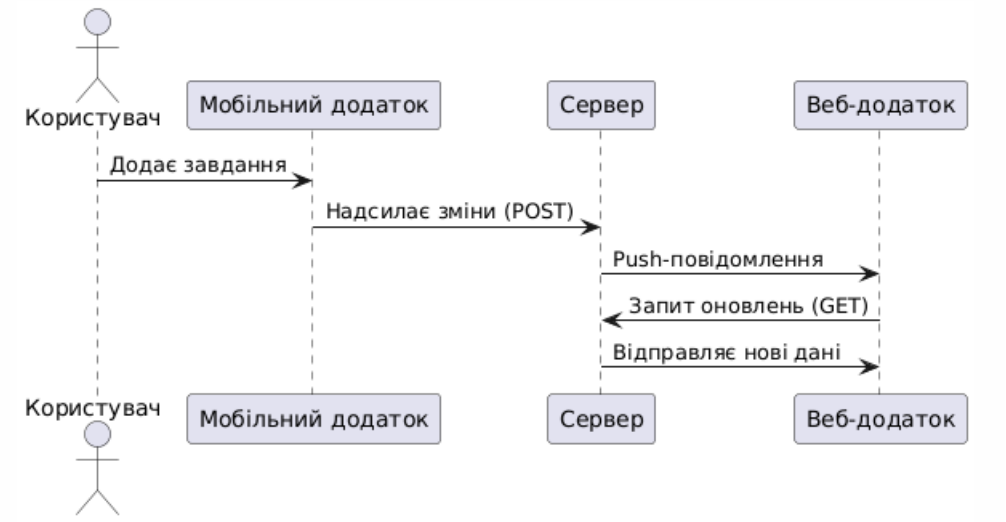

Рисунок 2. Діаграма послідовності

.Для складних сценаріїв з офлайн-редагуванням використовують спеціальні алгоритми вирішення конфліктів, такі як операційні трансформації або CRDT (Conflict-free Replicated Data Types).

З урахуванням різноманіття вимог до продуктивності, затримок, споживання ресурсів та надійності в умовах нестабільного з'єднання, вибір методу синхронізації є критично важливим при проєктуванні архітектури to-do застосунку. Наприклад, у простих задачах із невеликим навантаженням достатньо реалізувати Polling або Long Polling, проте для систем із високою частотою змін або великою кількістю одночасних користувачів доцільно використовувати WebSocket чи Push-нотифікації. Якщо ж система має працювати в офлайн-режимі з подальшою синхронізацією, особливо коли існує можливість конфлікту змін, тоді незамінними стають CRDT або подібні алгоритми.

Особливу увагу слід звернути на підтримку офлайн-режиму, адже користувачі to-do застосунків часто взаємодіють із системою у місцях із поганим покриттям або без підключення до мережі. У таких випадках критично важливо забезпечити можливість збереження змін локально з подальшою їх синхронізацією, щойно з’єднання буде відновлено. Це потребує впровадження локального кешування, механізмів контролю версій та обробки конфліктів.

Щоб узагальнити переваги, недоліки та особливості підтримки офлайн-режиму різних методів синхронізації, було наведено порівняльну таблицю (таблиця 1).

Таблиця 1. Порівняльна таблиця різних методів синхронізації

Метод	Переваги	Недоліки	Підхід до офлайн - роботи
Polling	Простота реалізації	Високе навантаження на сервер	Не підтримує

Long Polling	Менше запитів	Складність масштабування	Обмежена підтримка
WebSocket	Миттєва синхронізація	Високі вимоги до ресурсів	Вимагає додаткових рішень
Push-нотифікації	Енергоефективність	Залежність від сторонніх сервісів	Обмежена підтримка
CRDT	Автоматичне вирішення конфліктів	Складна реалізація	Повна підтримка

Як видно з таблиці, WebSocket є оптимальним для застосунків, де потрібна синхронізація в реальному часі (наприклад, спільне редагування завдань). Однак для офлайн-роботи краще підходять CRDT або гібридні рішення з локальним сховищем.

Сучасні платформи для розробки мобільних і веб-додатків пропонують цілий ряд інструментів для реалізації ефективної синхронізації даних. Популярні фреймворки, такі як Flutter, React Native або Angular, надають готові рішення для інтеграції з хмарними сервісами синхронізації. Зокрема, Firebase Realtime Database пропонує вбудовану підтримку синхронізації в реальному часі з автоматичним вирішенням конфліктів, що значно спрощує розробку to-do застосунків.

Важливим аспектом при виборі методу синхронізації є енергоефективність, особливо для мобільних пристроїв. Дослідження показують, що постійне використання WebSocket або часті polling-запити можуть значно знижувати час автономної роботи пристрою. Тому для мобільних застосунків часто використовують адаптивні стратегії, які змінюють частоту синхронізації залежно від рівня заряду батареї або типу мережевого з'єднання (Wi-Fi vs мобільний інтернет).

Проведене дослідження показало, що вибір оптимального методу синхронізації для кросплатформних to-do застосунків є багатокритеріальним завданням, яке потребує врахування різних факторів.

Найперспективнішими напрямками розвитку є комбіновані підходи, які поєднують переваги різних методів (наприклад, WebSocket для онлайн-режиму та CRDT для офлайн-роботи), а також використання спеціалізованих хмарних сервісів, що пропонують готові рішення для синхронізації. Майбутні дослідження в цій галузі можуть бути спрямовані на розробку більш ефективних алгоритмів вирішення конфліктів та оптимізацію витрат ресурсів при синхронізації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. WebSocket API. Опис API для двонаправленого зв’язку в реальному часі. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket (Дата звернення: 16.05.2025)
2. Push Notifications API. Технологія надсилання пуш-повідомлень користувачам. URL: https://developer.mozilla.org/en-US/docs/Web/API/Push_API (Дата звернення: 16.05.2025)
3. Offline Web Applications. Методики підтримки роботи додатків в офлайн-режимі. URL: https://developers.google.com/web/fundamentals/instant-and-offline/offline-cookbook (Дата звернення: 20.05.2025)
4. Synchronizing Offline Data. Приклади синхронізації офлайн-даних з сервером. URL: https://learn.microsoft.com/en-us/azure/architecture/example-scenario/mobile/offline-sync (Дата звернення: 24.05.2025)
5. CRDTs Explained. Пояснення принципів алгоритмів CRDT для синхронізації. URL: https://crdt.tech/ (Дата звернення: 24.05.2025)

6. Long Polling Technique. Опис методики довгого опитування серверу. URL: https://ably.com/concepts/long-polling (Дата звернення: 25.05.2025)

7. Polling vs WebSocket. Порівняння методів синхронізації в реальному часі. URL: https://ably.com/concepts/websockets-vs-polling (Дата звернення: 30.05.2025)

УДК 004.67

***Гейна Оксана**, здобучав ОПС фаховий молодший бакалавр*
IV курсу спеціальності «Комп'ютерні науки»
науковий керівник Кульчинська Н. З.

ДОСЛІДЖЕННЯ АЛГОРИТМІВ СИСТЕМИ РЕКОМЕНДАЦІЙ ДЛЯ РЕАЛІЗАЦІЇ ЕФЕКТИВНОГО МЕХАНІЗМУ ВЗАЄМОДІЇ З КОРИСТУВАЧЕМ

Рекомендаційна система – це алгоритм штучного інтелекту, зазвичай пов'язаний з машинним навчанням, який використовує великі дані, щоб запропонувати або порекомендувати споживачам додаткові продукти. Вони можуть ґрунтуватися на різних критеріях, включаючи минулі покупки, історію пошуку, демографічну інформацію та інші фактори. Рекомендаційні системи є дуже корисними, оскільки вони допомагають користувачам знаходити продукти та послуги, які вони, можливо, не знайшли б самостійно.

Системи рекомендацій навчені розуміти вподобання, попередні рішення та характеристики людей і продуктів, використовуючи дані, зібрані про їхню взаємодію. Сюди входять покази, кліки, вподобання та покупки. Завдяки своїй здатності передбачати інтереси та бажання споживачів на високо

персоналізованому рівні, рекомендаційні системи є фаворитами серед постачальників контенту та продуктів. Вони можуть спрямовувати споживачів до практично будь-якого продукту чи послуги, що їх цікавить.

Хоча існує величезна кількість алгоритмів і методів рекомендацій, більшість з них поділяються на такі широкі категорії: колаборативна фільтрація, фільтрація контенту і контекстна фільтрація.

Алгоритми колаборативної фільтрації рекомендують елементи на основі інформації про вподобання багатьох користувачів. Цей підхід використовує схожість поведінки користувачів, враховуючи попередні взаємодії між користувачами та об'єктами, алгоритми рекомендацій вчаться передбачати майбутню взаємодію. Ці рекомендаційні системи будують модель на основі минулої поведінки користувача, наприклад, товарів, придбаних раніше, або оцінок, наданих цим товарам, а також подібних рішень інших користувачів. Ідея полягає в тому, що якщо деякі люди приймали подібні рішення і робили покупки в минулому, наприклад, вибирали фільми, то існує висока ймовірність того, що вони зроблять однаковий вибір у майбутньому. Наприклад, якщо система колаборативної фільтрації виявляє, що два користувачі мають схожі літературні вподобання, вона може порекомендувати першому користувачеві твір, який уже сподобався другому. Для кращого розуміння принципу роботи фільтрації наведена схема на рисунку 1.

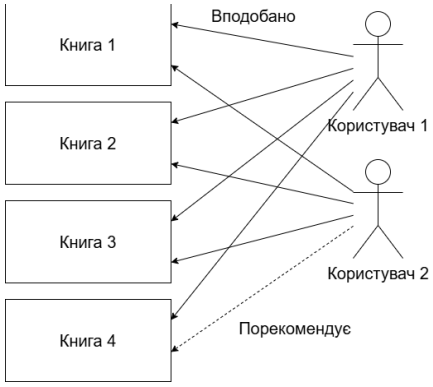

Рисунок 1 – Принцип роботи колаборативної фільтрації

На противагу цьому, контент-фільтрація використовує атрибути або особливості елемента, щоб рекомендувати інші елементи, схожі на вподобання користувача. Цей підхід ґрунтується на схожості характеристик товару та користувача, враховуючи інформацію про користувача та товари, з якими він взаємодіяв, а також його вподобання. Наприклад, якщо система рекомендацій бачить, що вам сподобалися певні твори, вона може запропонувати інші з подібним жанром, стилем або тематикою (рис. 2).

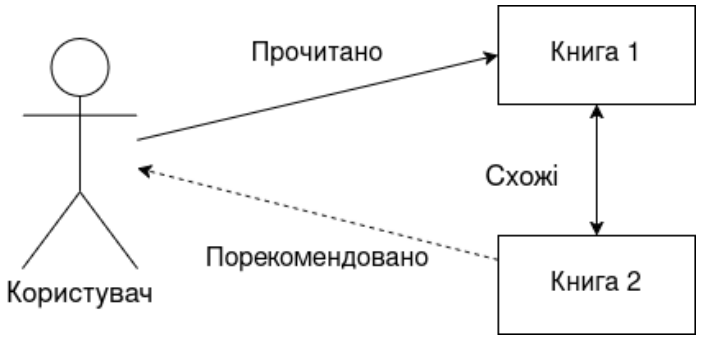

Рисунок 2 – Принцип роботи контентної фільтрації

Смаки та вподобання користувачів залежать від контексту. Місцезнаходження користувачів, люди, з якими вони перебувають, або час доби можуть впливати на вподобання чи потреби. Контекстно-орієнтовані рекомендації використовують цей контекст у процесі рекомендацій.

Вище розглянуті системи рекомендацій використовують двовимірні набори даних. Вони розглядають лише користувачів та елементи як вхідні дані для процесу рекомендацій. Контекстно-орієнтовані рекомендації також використовують інформацію про середовище для прогнозування вподобань користувачів. Таким чином, вони є багатовимірними.

Взаємодії в контекстно-орієнтованих рекомендаційних системах несуть більше інформації, ніж у традиційних підходах. Таким чином, більше не використовується бірграф для їх представлення. Користувачі, елементи та

контекст можуть бути представлені в n-частинному графі. Тут події пов'язані з контекстними елементами за допомогою багатьох зв'язків (рис.3).

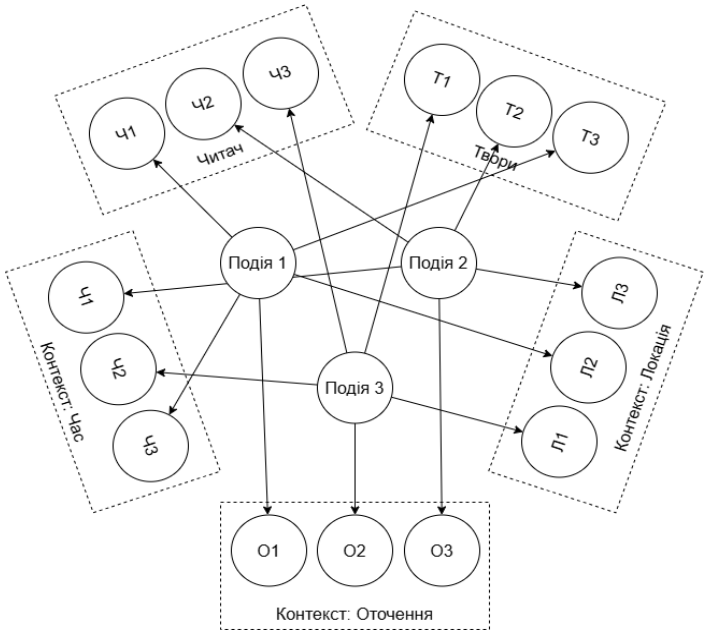

Рисунок 3 – Принцип роботи контекстно-орієнтованої фільтрації

Контекст може містити багато різних факторів, як-от час, погоду, місцезнаходження тощо, тому він часто буває дуже точним. Чим точніший контекст, тим складніше знайти інші події з точно такими ж умовами. Щоб це спростити використовують ієрархії – тобто впорядковують інформацію за рівнями загальності. Це дозволяє розширювати контекст і знаходити події з подібними, хоча й не ідентичними, умовами.

Контекстно-залежні рекомендації є точнішими, оскільки враховують більше факторів. Однак вони складні та стикаються з проблемою розрідженості даних.

Існують також гібридні методи, які намагаються використовувати як відомі метадані, так і набір спостережуваних взаємодій користувача з елементами. Цей підхід поєднує переваги методів фільтрації на основі вмісту та колаборативної фільтрації та дозволяє отримати найкращі результати.

В таблиці 1 наведено порівняння основних типів систем рекомендацій, що відрізняються за підходами до формування рекомендацій, необхідними даними, перевагами, недоліками та прикладами використання.

Таблиця 1 – Порівняння систем рекомендацій

Критерій	Колаборативна фільтрація	Фільтрація на основі вмісту	Контекстно-орієнтовані системи	Гібридні системи
Основна ідея	Рекомендації на основі схожих користувачів	Рекомендації на основі схожих об’єктів	Рекомендації з урахуванням середовища	Комбінація кількох підходів
Потрібні дані	Оцінки/взаємодії багатьох користувачів	Атрибути елементів та профіль користувача	Контекстні дані (час, місце, пристрій тощо)	Дані користувачів, об’єктів і контексту
Переваги	Висока персоналізація, не потребує знання атрибутів об'єктів	Добре працює для нових користувачів	Врахування специфіки ситуації	Зменшує недоліки окремих методів
Недоліки	Проблема "холодного старту", розрідженість даних	Не враховує думку інших, можливе перенавчання	Складність моделювання, проблема розрідженості	Висока складність реалізації
Приклади використання	Netflix, Amazon, Goodreads	Spotify, Google News	Uber, Foursquare	YouTube, TikTok

Бібліотеки / Інструменти	Surprise, LightFM	scikit-learn (TF-IDF, cosine), TensorFlow	CARSKit, Context-Aware Recommenders	LightFM (hybrid), TensorFlow Recommenders

Отже, кожен із розглянутих підходів до побудови систем рекомендацій має як переваги, так і обмеження, що визначають доцільність його застосування в конкретних умовах. Вибір оптимального методу залежить від специфіки задачі, доступності й якості даних, а також контексту використання системи. З огляду на це, гібридні рекомендаційні системи, які поєднують кілька підходів, демонструють високу гнучкість і результативність, оскільки дають змогу компенсувати недоліки окремих методів і забезпечують більш точні та персоналізовані результати.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is a Recommendation System?. *NVIDIA Data Science Glossary*. URL: https://www.nvidia.com/en-us/glossary/recommendation-system/ (дата звернення: 01.05.2025).
2. Casalegno F. Recommender Systems – A Complete Guide to Machine Learning Models. *Medium*. URL: https://medium.com/data-science/recommender-systems-a-complete-guide-to-machine-learning-models-96d3f94ea748 (дата звернення: 01.05.2025).
3. Panarin R. What You Need to Know Before Developing Recommender System. *Custom Software Development Company*. URL: https://maddevs.io/blog/recommender-system-using-machine-learning/ (дата звернення: 01.05.2025).

4. Context-aware and hybrid recommendations. *graphaware.com*. URL: https://graphaware.com/blog/content-aware-and-hybrid-recommendations/ (дата звернення: 01.05.2025).

5. 10 Remarkable Real World Examples Of Recommender Systems. AppsTek Corp. URL: https://appstekcorp.com/blog/10-remarkable-real-world-examples-of-recommender-systems/ (дата звернення: 15.05.2025).

УДК 004.4

***Давлетов Ренат**, здобувач ОПС фаховий молодший бакалавр*
IV курсу спеціальності «Комп'ютерні науки»
науковий керівник Івасьєв С.В.

РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ДЛЯ БЕЗПЕЧНОГО ОБМІНУ ДАНИМИ З КОРЕКЦІЄЮ ПОМИЛОК

Передача даних у цифровому вигляді є повсякденним явищем у діяльності як державних установ, так і бізнес-структур. Відкриті канали зв'язку, зокрема Інтернет або інші публічні мережі, часто стають об'єктом різноманітних атак. Зростання обсягів переданих даних супроводжується збільшенням кіберзагроз, можливістю перехоплення або спотворення інформації. Це зумовлює необхідність використання технологій шифрування для захисту інформації під час її передачі, а також застосування інструментів для забезпечення захисту від несанкціонованого доступу.

Актуальність розробки програмного засобу безпечної передачі інформації полягає в необхідності інтеграції методів криптографічного захисту та корекції помилок, що дозволяє одночасно вирішити дві ключові проблеми, зокрема забезпечити конфіденційність та цілісність інформації. У реальних

умовах передача даних повинна бути водночас швидкою і безпечною. Застосування систем шифрування дозволяє захистити інформацію, а коригуючі коди дозволяють виявляти і виправляти помилки, що можуть виникнути під час передавання даних.

Мета роботи полягає у розробці програмного засобу, що реалізує процес безпечної та надійної передачі даних, шляхом їх перетворення, шифрування та корекції помилок.

Для забезпечення цілісності даних запропоновано використання коригуючих властивостей коду Хеммінга [1], який дозволяє виявляти та виправляти помилки, що можуть виникнути під час передачі даних. Для забезпечення конфіденційності даних запропоновано застосування перетворення в ASCII-код, яке дозволяє представити текстові дані у числовій формі, що є зручним для подальшої математичної обробки.

Перетворення в ASCII також забезпечує універсальність проектованого програмного засобу, адже дозволяє підтримувати передачу як числових, так і текстових даних. ASCII є загальноприйнятим стандартом, що дозволяє без додаткових перетворень працювати з числовими значеннями в програмному середовищі. Завдяки цьому після декодування та корекції помилок вихідний текст можна безпомилково відновити.

Алгоритм роботи програмного засобу включає етапи шифрування та дешифрування даних (рисунок 1).

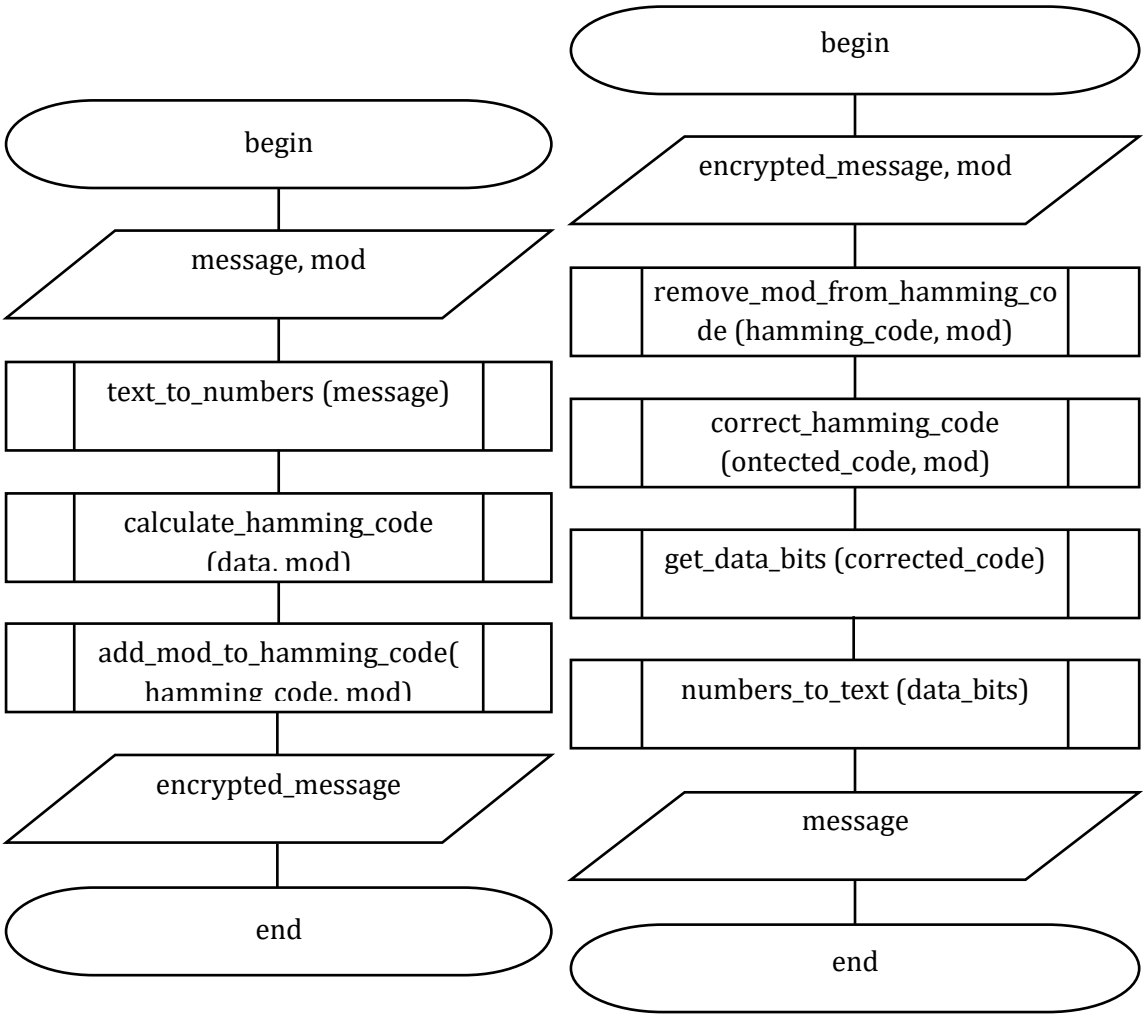

Рисунок 1 – Алгоритм шифрування та дешифрування даних

Основні кроки роботи алгоритму шифрування даних включають:

1. Отримання вихідних даних.
2. Перетворення вхідних даних у числовий формат.
3. Створення коду Хеммінга.
4. Модифікація кодованого повідомлення.
5. Виведення кодованого повідомлення.

Наведений алгоритм поєднує перетворення вхідних даних у числовий формат за допомогою ASCII-кодування, створення коду Хеммінга в системі числення з довільною основою p та використання операцій додавання за

модулем p для виявлення та виправлення помилок. Він дозволяє забезпечити необхідний рівень захисту для переданих даних, включаючи можливість виправлення одиничних помилок завдяки використанню коду Хеммінга та додаткового модулярного захисту.

Для коректного виправлення помилок важливо, щоб значення модуля p було не меншим ніж діапазон значень, які використовуються. Наприклад, для ASCII-кодів, які знаходяться в діапазоні від 0 до 127, необхідною є виконання умови $p>128$. Це гарантує, що всі значення ASCII будуть правильно зашифровані і відновлені без втрати інформації.

Програмний засіб (рисунок 2) реалізований за допомогою мови програмування Python [2], а його графічна частина - з використанням бібліотеки Tkinter [3], що дозволяє створювати легкий та швидкий інтерфейс, який не перевантажує систему і не потребує складної конфігурації. Tkinter використано як стандартну бібліотеку для створення GUI у Python, оскільки вона забезпечує мінімальні залежності, простоту інтеграції та кросплатформенність. Усі основні функції згруповані у логічно впорядковані блоки, що спрощує навігацію та використання програмного засобу. При зміні режиму (текстовий чи файловий) автоматично приховуються або відображаються відповідні поля вводу, кнопки та текстові підказки. Створений користувацький інтерфейс, який забезпечує ефективність роботи з програмним засобом і підвищує його доступність для кінцевого користувача.

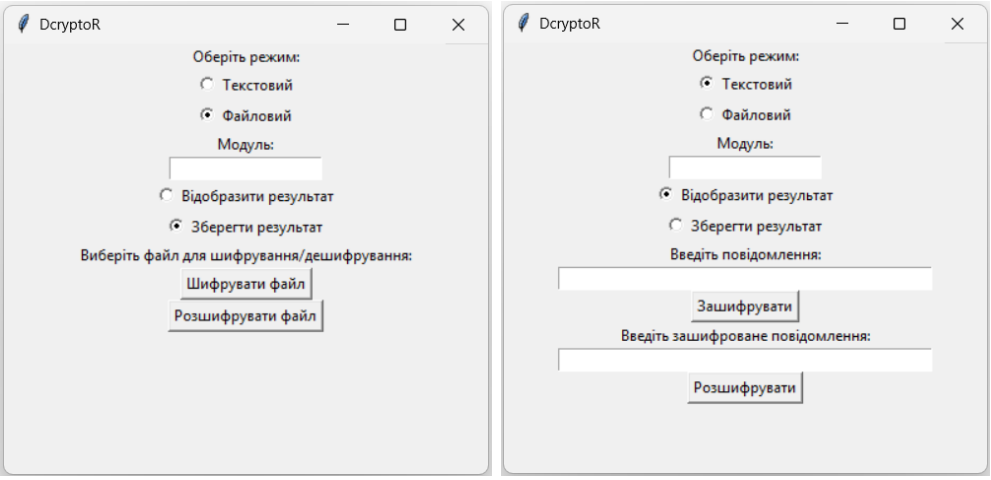

Рисунок 2 – Головне вікно програмного засобу

Для зручності використання кінцевими користувачами програмний засіб було зібрано у виконуваний файл формату .exe за допомогою інструмента PyInstaller, що дозволяє запускати програму без необхідності встановлення середовища Python. Це забезпечує швидкий доступ до функціоналу програмного засобу та спрощує його поширення серед користувачів як готовий до використання застосунок.

Проведено тестування розробленого програмного засобу з використанням різних сценаріїв, зокрема на повідомленнях різної довжини та із навмисним внесенням помилок у зашифровані дані. У всіх випадках алгоритм успішно виявляв та виправляв одиничні помилки, забезпечуючи коректне відновлення вихідного повідомлення.

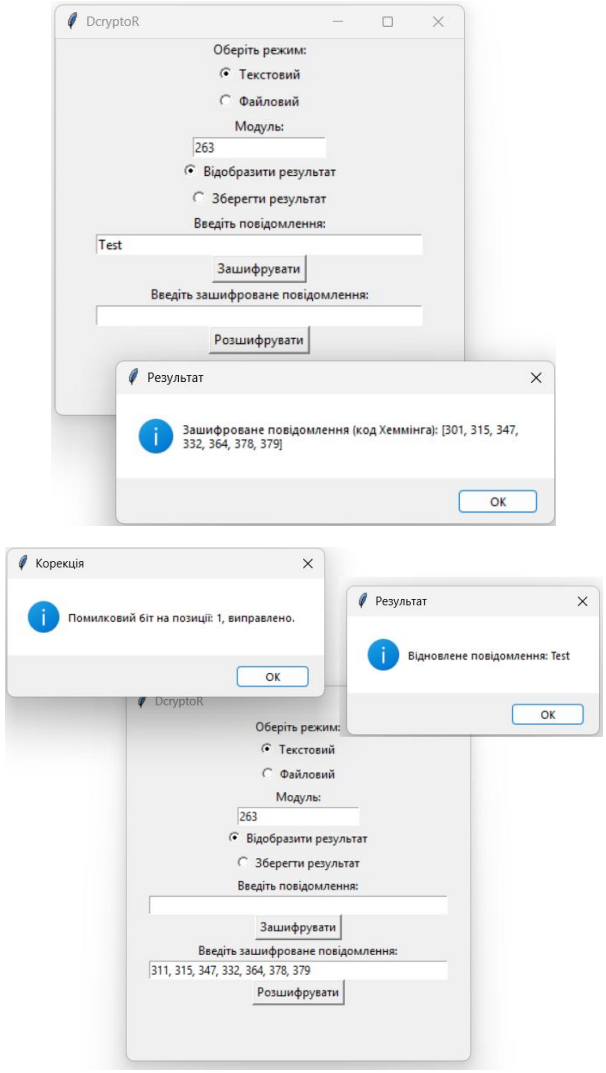

Рисунок 3 – Тестування програмного засобу

Отримані результати підтвердили ефективність і надійність реалізованого підходу щодо забезпечення конфіденційності та цілісності інформації під час її передачі. Під час тестування підтверджено працездатність графічного інтерфейсу користувача, який забезпечує зручну взаємодію з функціоналом програмного засобу, автоматично адаптується до вибраного режиму роботи та сприяє інтуїтивно зрозумілому керуванню процесами шифрування й декодування.

Постійне вдосконалення програмних засобів для шифрування та корекції помилок є необхідністю для забезпечення безпеки в умовах сучасних

кіберзагроз. Запропонований підхід дозволяє поєднати шифрування даних із механізмами корекції помилок, забезпечуючи високий рівень надійності при передачі. Розроблений програмний засіб забезпечує ефективну обробку текстової та файлової інформації у зручному для користувача середовищі. Реалізований інтерфейс та можливість використання без додаткових залежностей роблять програмний засіб доступним для широкого кола користувачів. Запропонована розробка дозволить забезпечити безпечний обмін даними у фінансових системах, корпоративних комунікаціях, а також зберігання конфіденційної інформації із можливістю відновлення після часткового пошкодження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Кодування сигналів в електронних системах / С.В. Денбновецький, І.В. Мельник, Л.Д. Писаренко ; КПІ ім. Ігоря Сікорського. –Київ : КПІ ім. Ігоря Сікорського, 2021. – 470с.

2. Python Software Foundation. Python Language Reference: вебсайт. URL: https://docs.python.org/3/reference/index.html: (дата звернення 22.05.2025).

3. Python Software Foundation. *Tkinter – Python interface to Tcl/Tk*. вебсайт. URL: https://docs.python.org/3/library/tkinter.html (дата звернення 22.05.2025).

УДК 004.89

***Кирич Олег**, здобувач вищої освіти*
IV курсу спеціальності 123 «Комп'ютерна інженерія»,
керівник роботиспеціаліст вищої категорії,
викладач-методист, Павлюс В.П.

АВТОМАТИЧНА СИСТЕМА КОНТРОЛЮ ДОСТУПУ: РОЗРОБКА ТА РЕАЛІЗАЦІЯ НА БАЗІ ТЕХНОЛОГІЙ ШТУЧНОГО ІНТЕЛЕКТУ ТА RFID

Що таке автоматична система контролю доступу? Це програмно-апаратний комплекс, який забезпечує керований доступ до об'єктів - шляхом ідентифікації осіб або транспорту. Основна мета - запобігти несанкціонованому потраплянню на територію.

Існує кілька основних способів ідентифікації в системах контролю доступу:

– RFID-картки, які легко інтегруються у будь-яку інфраструктуру;

– біометричні дані - відбитки пальців, розпізнавання обличчя або сітківки ока;

– сучасний напрям - з використанням штучного інтелекту наприклад автоматичне розпізнавання номерних знаків, яке дедалі частіше використовується в транспортній інфраструктурі.

Найбільш ефективним є поєднання кількох методів, що дозволяє підвищити надійність системи. Скажімо, якщо камера не змогла зчитати номер через бруд чи погане освітлення, в роботу вступає RFID-зчитувач або адміністратор через вебінтерфейс.

На ринку представлено безліч рішень. Наприклад, компанія Nedap спеціалізується на RFID-системах високого класу, які працюють на великих об’єктах. Їхні рішення - стабільні, інтегруються з відеоспостереженням, але вартість таких систем є досить високою.

ZKTeco пропонує біометричні термінали: розпізнавання обличчя, пальців, вен. Це дешевші, але менш стійкі до помилок у несприятливих умовах варіанти.

Компанія HID Global орієнтується на корпоративний рівень: використовує смарт-картки, мобільні ідентифікатори, і забезпечує багаторівневу аутентифікацію. Головний недолік - складність у налаштуванні.

Першими, хто заклав основу сучасних автоматичних систем контролю доступу, були науковці та розробники ще у 1970-х роках.

У 1973 році Чарльз Уолтон запатентував перший прототип RFID-картки, що стала фундаментом для майбутніх безконтактних ідентифікаторів.

У 1976–1979 роках у Великій Британії було розроблено перші системи автоматичного розпізнавання номерних знаків, які тоді працювали лише в умовах експериментального використання поліції.

У 1990-х з’являються перші промислові системи контролю доступу:

– HID Global зробив ставку на смарт-картки та багатофакторну автентифікацію;

– Nedap розпочав виробництво RFID-рішень для автоматичного проїзду транспорту через ворота;

– SKIDATA впровадила платформи контролю доступу для паркінгів і спортивних арен.

Проте навіть у сучасних реалізаціях цих систем існують суттєві обмеження:

– висока вартість обладнання та підписок;

– складність інтеграції зі стороннім ПЗ;

– залежність від хмарних серверів або ліцензійного середовища;

– закрита архітектура без доступу до вихідного коду.

Враховуючи ці недоліки, було вирішено створити альтернативну систему, яка мала б такі властивості:

– доступна для самостійного збирання та повторення;

– гнучка у налаштуванні під конкретний об’єкт;

– працює локально, без залежності від Інтернету;

– підтримує відкриті протоколи та розширення.

Результатом розробки стала система, побудована на базі мікроконтролера ESP32, яка включає такі елементи:

– Камера спостереження для фіксації транспорту та виявлення номерного знака;

– Модель Haar Cascade для визначення області номера та PaddleOCR для його розпізнавання;

– Модуль MFRC522 для зчитування RFID-карток як альтернативного або додаткового способу ідентифікації;

– Протокол MQTT для обміну повідомленнями між пристроями в режимі реального часу;

– Вебінтерфейс, реалізований на FastAPI з можливістю ручного керування, ведення журналу подій, управління базою даних;

– Локальна SQLite-база для зберігання користувачів, автотранспорту, подій.

Умови роботи системи такі:

– рекомендована камерна роздільна здатність - не менше 720p для стабільного розпізнавання номерів;

– оптимальна відстань зчитування номерного знаку - 2-4 метри при прямому куті огляду;

– дальність дії RFID-зчитувача - до 60 мм (при роботі з MIFARE-картами);

– підключення ESP32 здійснюється через WiFi, живлення - 5 В, рекомендовано використовувати стабілізований блок живлення;

– система працює в мережі, не потребує доступу до Інтернету, що підвищує рівень безпеки;

– можлива робота при температурі від -10°C до +45°C, за умови герметизації камери та зчитувача;

– час реакції від ідентифікації до відкриття воріт - менше 1 секунди.

Таким чином, користувач має змогу:

– автоматично ідентифікувати автомобіль за номерним знаком;

– використовувати RFID-картку у разі відмови відеоаналізу;

– переглядати події, реєструвати нові автомобілі та керувати доступом через вебпанель;

– масштабувати систему: додати нові зчитувачі, камери, користувачів або інтегрувати її з сигналізацією.

– Це рішення поєднує доступність, функціональність та незалежність від сторонніх сервісів - те, чого не вистачає багатьом комерційним аналогам.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Tech Time Warp: The father of RFID technology. SmarterMSP. Вебсайт. URL: https://smartermsp.com/tech-time-warp-the-father-of-rfid-technology/?utm_source=chatgpt.com (дата звернення:. 07.05.2025).

2. Automatic number-plate recognition. Wikipedia. Вебсайт. URL: https://en.wikipedia.org/wiki/Automatic_number-plate_recognition?utm_source=chatgpt.com (дата звернення:. 07.05.2025).

3. Charles Walton. LEMELS N-MIT. Вебсайт. URL: https://lemelson.mit.edu/resources/charles-walton?utm_source=chatgpt.com (дата звернення:. 07.05.2025).

4. ZKTeco Official. Вебсайт.– https://zkteco.eu (дата звернення:. 07.05.2025).

5. Nedap Identification Systems. Вебсайт. URL: https://www.nedapidentification.com (дата звернення:. 07.05.2025).

6. HID Global. Вебсайт. URL: https://www.hidglobal.com (дата звернення:. 07.05.2025).

УДК 004.056

***Кульчинська Валентина**, здобувач фахової передвищої освіти IV курсу спеціальності «Комп'ютерні науки»*
науковий керівник Сиротюк О.Б.

АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ WEB-БАЗОВАНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

Web-базована інформаційна система (ІС) – це програмно-апаратний комплекс, що працює в середовищі Інтернету або корпоративної мережі, призначений для автоматизації процесів обробки, зберігання та надання інформації користувачам через веб-інтерфейс.

Типова структура web-ІС включає три рівні:

1. Презентаційний рівень (клієнтська частина) – реалізується з використанням HTML, CSS, JavaScript та сучасних фреймворків (React, Vue, Bootstrap тощо). Основне завдання – забезпечити зручний та інтуїтивно зрозумілий користувацький інтерфейс.

2. Логічний рівень (серверна частина) – відповідає за бізнес-логіку, обробку запитів, авторизацію користувачів, взаємодію з базою даних. Тут застосовуються серверні мови програмування (PHP, Python, JavaScript/Node.js) та фреймворки (Laravel, Django, Express).

3. Рівень даних (СУБД) – реалізує зберігання структурованої інформації, забезпечує доступ до неї, дозволяє виконувати запити, сортування, фільтрацію, аналітику.

У випадку онлайн-системи для продажу товарів висуваються додаткові вимоги:

- підтримка великої кількості транзакцій;
- захист персональних даних;
- обробка платежів;
- можливість масштабування.

У процесі вибору СУБД для онлайн-системи важливо враховувати тип проекту, обсяг даних, вимоги до швидкодії, підтримку транзакцій, наявність технічної підтримки, простоту інтеграції з іншими компонентами (таблиця 1).

Таблиця 1 – Порівняння найпопулярніших СУБД для реалізації web-систем

Параметр	MySQL	PostgreSQL	SQLite	MongoDB
Тип	Реляційна	Реляційна	Локальна	NoSQL (документальна)
Продуктивність	Висока	Висока	Обмежена	Висока

Масштабованість	Висока	Висока	Низька	Дуже висока
Транзакції	Так (InnoDB)	Так	Обмежено	Ні
Простота	Легка	Середня	Дуже проста	Складна для SQL-зв’язків
Сумісність	PHP, Python	PHP, Python	Усі	Node.js, Python
Ліцензія	GPL	PostgreSQL	Public Domain	SSPL

MySQL пропонує широкий спектр функціональних можливостей, які забезпечують її численні переваги. Ця СУБД вирізняється багатим набором інструментів, що роблять її особливо привабливою для використання при розробці web-систем:

- стабільна та перевірена роками технологія, що підтримує всі типові задачі онлайн-магазину;
- широка підтримка хостингів – більшість провайдерів вже мають встановлений MySQL;
- велика спільнота та документація – спрощує розв’язання технічних проблем;
- підтримка реляційної моделі та транзакцій (через InnoDB);
- гнучка інтеграція з PHP та популярними фреймворками, наприклад, Laravel, Yii, Symfony.

Вибір інструментів для взаємодії із системою управління базами даних MySQL має вирішальне значення для ефективної реалізації веб-системи. До таких засобів належать:

- мови програмування, які забезпечують обробку логіки на серверній частині;
- фреймворки та бібліотеки, які спрощують розробку, структурують код і забезпечують засоби безпеки;
- ORM-системи (Object-Relational Mapping), які полегшують роботу з базою даних через об’єктну модель;
- інструменти адміністрування, які дають змогу керувати БД без глибоких знань SQL.

PHP – найпоширеніша серверна мова для створення веб-систем, яка історично тісно інтегрована з MySQL. Переваги:

- вбудовані функції роботи з MySQL (mysqli, PDO);
- підтримка популярних фрейморків, таких як Laravel, Symfony, WordPress – усі працюють із MySQL;
- легка інтеграція з HTML та JavaScript;
- широка спільнота та багато навчальних ресурсів.

Недоліками є складність у забезпеченні безпеки та масштабованості (MVC, фреймворки).

Python з фреймворком Django – також потужне середовище для швидкої розробки веб-додатків. Перевагами є висока читабельність коду, вбудований ORM, який легко підключається до MySQL, активна підтримка спільноти. Та при виборі даної мови слід врахувати такі недоліки як менша популярність для хостингу в порівнянні з PHP, більша початкова складність налаштування.

Node.js дозволяє реалізовувати як клієнтську, так і серверну частину за допомогою JavaScript. Для взаємодії з MySQL використовують бібліотеки mysql2, sequelize та фреймворк Express.js для створення REST API.

Перевагами є асинхронність, масштабованість, сучасний стек (MERN/MEVN). Але вагомим недоліком є більша складність обробки помилок та логіки у порівнянні з PHP.

Фреймворки використовують для спрощення, прискорення та структурування процесу розробки програмного забезпечення, зокрема веб-додатків. Вони надають готовий каркас (framework), в межах якого розробник створює власну логіку, не починаючи все "з нуля".

Основні цілі використання фреймворків:

- швидкість розробки – фреймворки надають набір готових функцій (робота з базами даних, маршрутизація, автентифікація тощо), що значно скорочує час на написання коду;
- стандартизація коду – вони забезпечують дотримання загальноприйнятих підходів та структур (наприклад, MVC – Model-View-Controller), що робить код більш читабельним і зрозумілим для інших розробників;
- безпека – більшість сучасних фреймворків мають вбудовані засоби захисту від поширених атак: SQL-ін’єкцій, XSS, CSRF тощо;
- підтримка та масштабованість – завдяки своїй структурі, фреймворки дозволяють легше підтримувати та розширювати проєкт у майбутньому;
- зменшення кількості помилок – використання вже перевірених компонентів знижує ризик виникнення критичних помилок у коді.
- інтеграція з іншими технологіями – багато фреймворків легко інтегруються з бібліотеками, API, сторонніми сервісами, що розширює функціональні можливості проєкту.

Laravel є найпопулярнішим фреймворком у світі PHP. Має потужну інтеграцію з MySQL і забезпечує:

- ORM Eloquent для об’єктної роботи з таблицями;
- міграції та seeders для управління схемою БД;
- Blade шаблони для відображення даних;
- готову систему автентифікації та авторизації;
- підтримку кешування, черг, веб-сокетів.

Недоліки: потребує певного рівня підготовки, особливо у роботі з міграціями та сервісами.

Фреймворк Django дозволяє швидко створювати надійні веб-додатки з підтримкою MySQL. Основні функції:

- ORM для моделювання БД через класи Python;
- адмін-панель за замовчуванням;
- вбудовані засоби безпеки (XSS, CSRF, SQL-захист).

Але важливим недоліком є складність налаштування проєкту на продакшн-сервері.

Express.js – мінімалістичний фреймворк для Node.js, який забезпечує створення API для фронтенду. Використовується з MySQL через mysql2 (низькорівнева бібліотека для виконання SQL-запитів) та sequelize (ORM-аналог Eloquent). Надає такі переваги як асинхронна обробка, висока продуктивність; при цьому недоліками є велика кількість налаштувань, відсутність "з коробки" рішень для авторизації, валідації.

Інструменти адміністрування MySQL використовуються для керування, налаштування, моніторингу та оптимізації роботи бази даних без необхідності постійно писати SQL-запити вручну. Вони значно полегшують життя як розробникам, так і адміністраторам баз даних (DBA):

- створення та редагування баз даних і таблиць з використанням візуального інтерфейсу (дозволяє створювати таблиці, визначати поля, типи даних, індекси без написання SQL-коду);

- виконання SQL-запитів, виведення результатів у зручній формі;
- імпорт та експорт даних між різними серверами, створення резервних копій (наприклад, у форматі SQL або CSV);
- керування користувачами та правами доступу (додавання нових користувачів, встановлення паролів, обмеження доступу до окремих баз або таблиць);
- аналіз навантаження, кількості підключень, блокувань, часу виконання запитів тощо;
-
- візуальне моделювання бази даних – створення ER-діаграм (Entity-Relationship), що особливо корисно при проєктуванні структури БД.

Найпоширенішими інструментами адміністрування MySQL на сьогоднішній день є phpMyAdmin та MySQL Workbench.

PhpMyAdmin надає безкоштовний веб-інтерфейс, забезпечує роботу з таблицями, SQL-запитами, імпорт/експорт, також тут можна швидко створити або редагувати схему бази.

Щодо MySQL Workbench, то він є більш потужним, ніж phpMyAdmin, також підтримує візуальне моделювання БД (ER-діаграми) та автоматизацію резервного копіювання та адміністрування.

В контексті розробки онлайн-системи найдоцільнішим вибором є стек Laravel + MySQL. Laravel надає розробнику інструменти, які прискорюють процес створення системи, забезпечують безпечну роботу з даними, а також дозволяють швидко масштабувати проєкт. MySQL у поєднанні з Laravel та Eloquent ORM формує стабільну основу для надійної web-базованої інформаційної системи.

Перевагами обраного підходу є:

1. Продуктивність – MySQL обробляє тисячі запитів на хвилину, що підходить для онлайн-магазинів середнього та великого розміру.
2. Безпека – Laravel забезпечує захист від основних атак (XSS, CSRF, SQL-ін’єкції).
3. Гнучкість – легке масштабування, зміна схеми БД через міграції.
4. Швидка розробка – Laravel спрощує реалізацію автентифікації, маршрутизації, обробки форм.
5. Масштабованість – структура дозволяє легко додавати нові функції (API, оплати, логістика).
6. Легкість у розгортанні – MySQL та Laravel мають підтримку більшості хостинг-платформ.

Отже, в процесі проведеного аналізу було розглянуто сучасні підходи до розробки web-систем, особливу увагу приділено вибору системи керування базами даних (СУБД) та засобам її інтеграції з веб-застосунками.

Серед різноманіття СУБД, MySQL було обрано як оптимальний варіант для побудови інформаційної системи онлайн-магазину. Це рішення зумовлене її високою продуктивністю, підтримкою транзакцій, масштабованістю, сумісністю з популярними хостинг-платформами.

Також було проведено аналіз інструментів веб-розробки, які забезпечують ефективну взаємодію з MySQL. Особливу увагу приділено фреймворку Laravel, що реалізує архітектуру MVC, надає засоби безпеки, ORM (Eloquent), підтримку REST API та зручний механізм шаблонізації.

У результаті дослідження було спроєктовано web-систему для продажу товарів, що забезпечує основні бізнес-функції – управління товарами, замовленнями, користувачами та адміністративною панеллю.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Глушаков І. С., Мельник С. П. *Web-програмування: навчальний посібник*. Київ: Ліра-К, 2021. 264 с.

2. MySQL Documentation. *MySQL 8.0 Reference Manual: вебсайт*. URL: https://dev.mysql.com/doc/refman/8.0/en/ (дата звернення: 27.05.2025).

3. Laravel Documentation. *The PHP Framework for Web Artisans: вебсайт*. URL: https://laravel.com/docs (дата звернення: 27.05.2025).

4. Селко Джо. *SQL для професіоналів. Рецепти програмування.* Дніпро: Баланс Бізнес Букс, 2015. 448 с.

5. Кузнецов С. М., Шевченко В. І. *Системи управління базами даних: навчальний посібник.* Харків: ХНУРЕ, 2020. 198 с.

6. phpMyAdmin Documentation. *Офіційна документація: вебсайт*. URL: https://www.phpmyadmin.net/docs/ (дата звернення: 27.05.2025).

УДК 004.42

***Литвинчук Аліна**, здобувач ОКР фаховий молодший бакалавр*
IV курсу спеціальності «Комп'ютерні науки»
науковий керівник Гавришків Н. Г.

РЕАЛІЗАЦІЯ АСИНХРОННОЇ ВЗАЄМОДІЇ КЛІЄНТА З RESTFUL API СЕРВЕРОМ У КРОСПЛАТФОРМЕНОМУ СЕРЕДОВИЩІ FLUTTER

Асинхронність — це процес обробки введення/виводу, що дозволяє продовжити обробку інших завдань, не чекаючи завершення попереднього завдання.

У синхронній моделі виконання коду кожна операція послідовно очікує завершення попередньої, що може призводити до критичних затримок у роботі всієї програми, коли певна команда вимагає тривалого часу на виконання.

Асинхронне програмування вирішує цю проблему шляхом виведення потенційно довготривалих операцій з основного потоку виконання, запобігаючи його блокуванню та дозволяючи програмі паралельно виконувати інші завдання.

Цей підхід надзвичайно ефективний для вирішення різноманітних прикладних задач. Одним із найбільш цінних аспектів асинхронності є можливість користувача продовжувати взаємодію з додатком під час виконання фонових операцій.

Практичний приклад: студент хоче переглянути свої оцінки в мобільному застосунку, після того як він обрав категорію оцінок, надсилається відповідний запит до сервера, де відбувається його обробка, що може зайняти значний час. При використанні синхронного методу інтерфейс залишився б заблокованим, що значно погіршує користувацький досвід.

Впровадження асинхронності в такому випадку забезпечує розгалуження основного потоку виконання на дві незалежні гілки: одна продовжує підтримувати інтерактивність користувацького інтерфейсу, в той час як інша займається обробкою мережевого запиту та очікуванням відповіді від сервера. Це гарантує плавність і безперервність роботи додатку навіть під час виконання ресурсомістких операцій.

Сучасне асинхронне програмування реалізується через три основні архітектурні підходи, кожен з яких має свою специфіку застосування відповідно до характеру вирішуваних завдань (рис. 1).

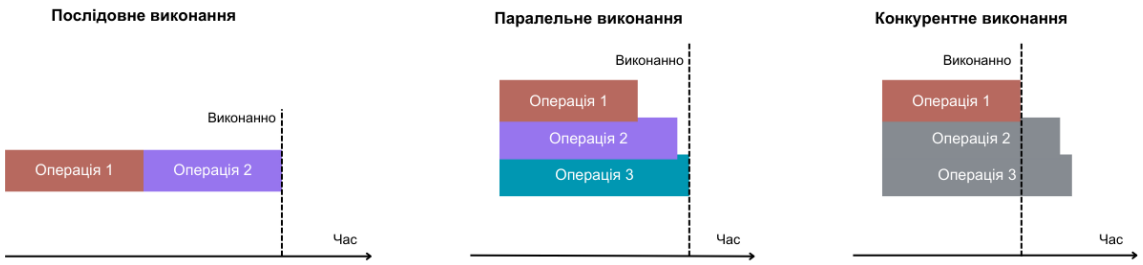

Рисунок 1 – Шаблони асинхронності

– послідовне виконання оптимальне для взаємозалежних операцій, де результат кожної наступної дії спирається на дані, отримані попередньою;

– паралельне виконання найбільш ефективне у сценаріях з множинними незалежними операціями, коли критично важливо отримати результати всіх запущених процесів;

– конкурентне виконання підходить для ситуацій, коли проводиться декілька однотипних незалежних операцій, і достатньо успішного виконання хоча б однієї з них.

REST API (Representational State Transfer) — це архітектурний підхід до створення веб-сервісів, який базується на принципах та обмеженнях протоколу HTTP. Це не просто набір технологій, а сукупність архітектурних принципів, які дозволяють створювати масштабовані, надійні та передбачувані веб-сервіси. У контексті розробки мобільних додатків на Flutter, REST API слугує стандартизованим інтерфейсом комунікації між клієнтським додатком та серверною частиною.

Для здійснення HTTP запитів у Flutter-додатках доступні різні бібліотеки, кожна з яких має свої особливості та пріоритетні сфери застосування.

HTTP Package (package:http) — це мінімалістичне рішення від Dart Team, що ідеально підходить для базових HTTP-запитів. Основні переваги

включають простоту використання з нульовими залежностями, офіційну підтримку та повну сумісність з усіма платформами Flutter. Пакет має нативну підтримку async/await синтаксису Dart, що робить код зрозумілим та читабельним. Однак його обмежена функціональність вимагає ручної реалізації багатьох речей: серіалізації JSON через jsonDecode(), перевірки HTTP-помилок (які не генерують винятки автоматично), а також самостійного додавання інтерсепторів, скасування запитів та таймаутів.

Dio Package (package:dio) — це потужна альтернатива з розширеними можливостями для складних HTTP-сценаріїв. Пакет пропонує вбудовані інтерсептори для глобальної обробки запитів, автоматичну серіалізацію JSON, скасування запитів через CancelToken, відстеження прогресу завантаження файлів та гнучку конфігурацію з базовими URL і таймаутами. Додатково доступна екосистема плагінів для кешування та повторних запитів. Недоліками є складність для початківців та ризик надмірної інженерії при використанні для простих задач, де базового HTTP-пакету було б достатньо.

Важливим критерієм під час вибору бібліотеки для реалізації є важливість інтерцептора.

Інтерцептор (interceptor) — це спеціальний механізм, який перехоплює HTTP-запити та відповіді до їх надсилання або після отримання, щоб автоматично обробити або модифікувати їх. У Flutter він зазвичай реалізується через бібліотеку Dio, яка дозволяє інтегрувати інтерцептори у ланцюжок запитів, саме тому було обрано її.

Асинхронність у Dart реалізується за допомогою конструкцій Future, async та await, що дозволяють виконувати тривалі операції — зокрема, мережеві запити — без блокування основного потоку UI.

Future<T> — це "обіцянка" результату, який буде отримано в майбутньому після завершення асинхронної операції, такої як HTTP-запит або

читання файлу. Future представляє об'єкт, який поки не має конкретного значення, але матиме його згодом. Це дозволяє "підписатися" на майбутній результат і обробити його, коли асинхронна операція завершиться, без блокування виконання програми.

async — це ключове слово для оголошення асинхронної функції, яка повертає Future. Воно сигналізує компілятору, що всередині функції можуть використовуватись операції await для роботи з асинхронним кодом. async перетворює звичайну функцію на асинхронну, дозволяючи писати читабельний код для операцій, які потребують часу на виконання.

await використовується перед викликом Future-функції для очікування завершення асинхронної операції. Воно призупиняє виконання поточної функції до отримання результату, але при цьому не блокує UI-потік додатку. await дозволяє писати асинхронний код, який виглядає та читається як звичайний синхронний код, роблячи логіку програми більш зрозумілою.

Реалізовано цей підхід в кросплатформеному застосунку “е-Студент”, який є сучасним рішенням для організації навчального процесу в Галицькому фаховому коледжі, він дозволяє студентам та викладачам зручно переглядати персональний розклад занять, оцінки, відвідуваність, тощо.

Було обрано послідовний шаблон асинхронності. Для взаємодії із серверною частиною реалізовано клас «RestAPIClient», у якому визначені методи для авторизації, отримання інформації про користувача та завантаження оцінок. Кожен з цих методів використовує await, щоб дочекатися результату від сервера, не блокуючи при цьому основний потік додатку. Усі методи повертають об'єкти Future, які дозволяють UI "дочекатися" даних без заморожування інтерфейсу.

Також реалізовано «AuthInterceptor», який забезпечує централізоване керування токенами автентифікації в Dio клієнті, автоматично додаючи

access_token до кожного запиту та перевіряючи його термін дії. При виявленні протермінованого токена інтерсептор автоматично виконує його оновлення через refresh механізм і повторює початковий запит, а у випадку невдалого оновлення — очищує сесію користувача. Такий підхід дозволяє уникнути дублювання логіки авторизації в кожному API-методі, централізуючи всю обробку токенів в одному місці.

Отримані з сервера дані (наприклад, ім’я, прізвище, оцінки) асинхронно зберігаються в безпечному сховищі «SecureStorage», що дозволяє використовувати їх в інтерфейсі навіть після перезапуску додатку.

Використання конструкцій Future, async і await забезпечує плавну роботу додатку без затримки UI, а централізоване керування токенами автентифікації через інтерцептор дозволяє уникнути дублювання коду. Результатом стала ефективна система взаємодії з серверною частиною навчального застосунку з автоматичним збереженням даних у SecureStorage.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Asynchronous Programming. URL: https://www.indeed.com/career-advice/career-development/asynchronous-programming (дата звернення: 10.05.2025).

2. A deep dive into asynchronous patterns. URL: https://abinavravi.medium.com/a-deep-dive-into-asynchronous-patterns-e0379905fdaa (дата звернення: 10.05.2025).

3. Deep Dive into Popular Flutter Packages: Dio. URL: https://ms3byoussef.medium.com/deep-dive-into-popular-flutter-packages-dio-09e9aea86df8 (дата звернення: 10.05.2025).

4. Efficiently Handling HTTP Requests in Flutter with http Package. URL: https://medium.com/@ugamakelechi501/efficiently-handling-http-requests-in-flutter-with-http-package-fb719d15deb2 (дата звернення: 10.05.2025).

5. Interceptor in Flutter. URL: https://medium.com/@manjeetkushalmallick/interceptor-in-flutter-9f788f3c90f6 (дата звернення: 10.05.2025).

6. Asynchronous programming: futures, async, await. URL: https://dart.dev/libraries/async/async-await (дата звернення: 10.05.2025).

УДК 004

***Марчук Арсен**, здобувач ОКР фаховий молодший бакалавр*
IV курсу спеціальності «Комп'ютерні науки»
науковий керівник Кульчинська Н.З.

ДОСЛІДЖЕННЯ МЕХАНІЗМІВ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ТА КОНФІДЕНЦІЙНОСТІ ДАНИХ КОРИСТУВАЧІВ У МОБІЛЬНИХ ЗАСТОСУНКАХ

Стрімкий розвиток мобільних технологій супроводжується зростанням загроз безпеці даних, що обробляються мобільними застосунками. Збільшення обсягів персональних даних, які щодня обробляються мільярдами застосунків, робить захист інформації критично важливим завданням. Ця робота присвячена аналізу основних загроз безпеці даних у мобільних застосунках, дослідженню механізмів їх захисту, таких як шифрування, автентифікація, управління доступом, захист від шкідливого програмного забезпечення та освіта користувачів, а також розробці рекомендацій для підвищення безпеки. Об'єктом дослідження є мобільні застосунки, що працюють з персональними

даними користувачів, а предметом — технології та методи забезпечення їхньої безпеки й конфіденційності. Використані методи включають теоретичний аналіз, індукцію, дедукцію та аналіз практичних кейсів. Наукова новизна полягає в систематизації сучасних методів захисту даних, оцінці їхньої ефективності та адаптації до нових кіберзагроз.

Мобільні застосунки є вразливими до широкого спектра загроз, які загрожують конфіденційності, цілісності та доступності даних [1]. Основні загрози включають:

– Шкідливе програмне забезпечення (Malware): Віруси, троянські програми, шпигунське ПЗ (spyware) та рекламні модулі (adware) можуть викрадати персональні дані, відстежувати дії користувача або генерувати несанкціоновані підписки [2]. Наприклад, у 2019 році троян "Joker" у Google Play підключав користувачів до платних сервісів без їхнього відома, викрадаючи SMS-повідомлення та контакти [12].

– Фішинг: Шахрайські методи, спрямовані на отримання конфіденційної інформації (логіни, паролі, банківські дані) через підроблені повідомлення, електронні листи чи сайти. У 2019 році фішингові атаки на Instagram використовували фальшиві повідомлення від служби підтримки, що призводило до викрадення акаунтів і поширення спаму [5].

– Атаки через незахищені Wi-Fi мережі: Атаки типу "людина посередині" (MITM) дозволяють перехоплювати дані в публічних мережах. Зловмисники створюють фальшиві точки доступу, щоб отримати паролі чи банківські реквізити [7].

– Небезпечні API: Вразливості в API дозволяють несанкціонований доступ до баз даних [8]. У 2018 році через недостатню безпеку API Facebook зловмисники отримали доступ до даних 50 мільйонів користувачів, включаючи фотографії та приватні повідомлення [13].

– Недостатня безпека зберігання даних: Незашифровані дані на пристроях вразливі при фізичному доступі чи через системні уразливості. У 2020 році TikTok зберігав відео та геолокаційні дані без належного шифрування, що призвело до витоків [12].

– Ненадійні паролі та автентифікація: Слабкі або повторно використані паролі полегшують компрометацію. У 2016 році через вразливість API Uber хакери отримали доступ до даних 57 мільйонів користувачів, включаючи інформацію про поїздки та платіжні дані [10].

– Підключення до незахищених серверів: Вразливі сервери можуть бути використані для несанкціонованих операцій. Застаріле серверне ПЗ збільшує ризик атак [9]. Для узагальнення даних типові кіберзагрози представлено в таблиці 1

Таблиця 1 – Таблиця кіберзагроз та прикладів їх реалізації

Загроза	Опис	Приклад
Шкідливе ПЗ	Викрадення даних, несанкціоновані дії	Троян "Joker" (Google Play)
Фішинг	Обман для отримання даних	Атаки на Instagram (2019)
Незащищені Wi-Fi	Перехоплення даних (MITM)	Фальшиві точки доступу
Небезпечні API	Несанкціонований доступ до баз	Витік даних Facebook (2018)
Недостатнє шифрування	Витік даних при фізичному доступі	TikTok (2020)
Ненадійні паролі	Компрометація через слабкі паролі	Витік даних Uber (2016)

Загрози поділяються за походженням, характером впливу, об’єктом та рівнем впливу [1]:

– За походженням: Природні (стихійні лиха, технічні збої) та штучні (хакерські атаки, помилки персоналу).

– За характером: Пасивні (прослуховування трафіку) та активні (DDoS-атаки, віруси).

– За об’єктом: Апаратне забезпечення (фізичне пошкодження), програмне забезпечення (експлуатація вразливостей), дані (порушення конфіденційності, цілісності, доступності).

– За рівнем впливу: Високий (зупинка бізнес-процесів), середній (тимчасові збої), низький (незначні незручності).

Для протидії загрозам застосовуються комплексні механізми захисту, що охоплюють шифрування, автентифікацію, управління доступом, захист від шкідливого ПЗ та освіту користувачів.

Шифрування даних. Шифрування забезпечує захист даних шляхом їх перетворення у формат, доступний лише з ключем. Симетричне шифрування (AES) ефективне для великих обсягів даних, асиметричне (RSA, ECC) — для обміну ключами. Гібридне шифрування, як у HTTPS, комбінує обидва методи [15]. Повнодискове шифрування захищає дані на пристроях, а SSL/TLS — під час передачі [9]. Управління ключами є критично важливим, оскільки їх втрата унеможливлює доступ до даних. З появою квантових комп’ютерів розвивається постквантова криптографія [14]. Структуру основних типів шифрування подано на рисунку 1.

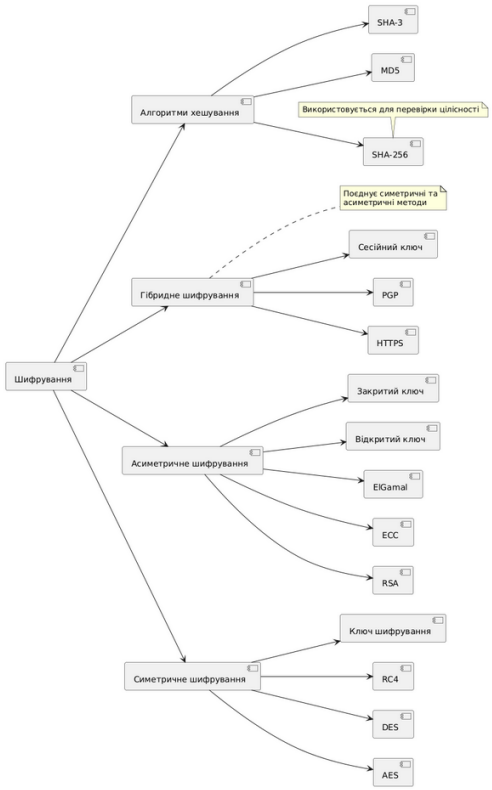

Рисунок 1 – Діаграма компонентів шифрування

Автентифікація. Механізми автентифікації перевіряють особу користувача. Базова автентифікація використовує логін і пароль, але її слабкість — у людському факторі (слабкі паролі, їх повторне використання) [10]. Двофакторна автентифікація (2FA) додає другий фактор (SMS-код, токен), а багатофакторна (MFA) включає біометрію, геолокацію чи поведінкові патерни. Однократна автентифікація (SSO) з SAML чи OAuth забезпечує доступ до кількох систем після одного входу. Безпаролева автентифікація (FIDO2, WebAuthn) набирає популярності [6]. Структуру механізмів автентифікації наведено на рисунку 2.

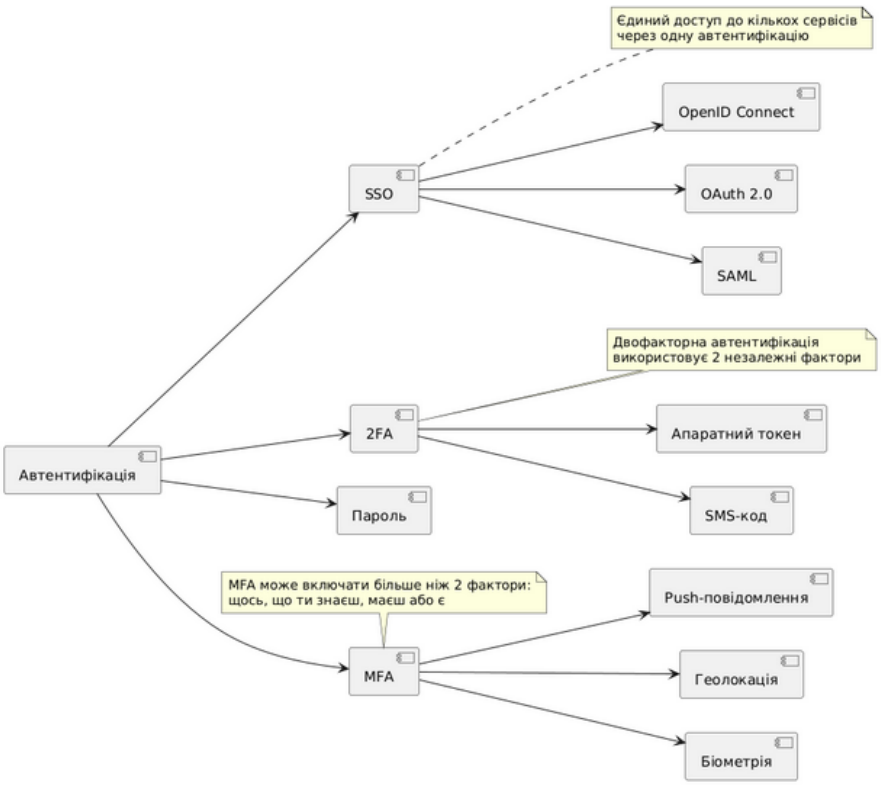

Рисунок 2 – Діаграма компонентів системи автентифікації

Після автентифікації система авторизації визначає права користувача за ролями (адміністратор, користувач). Сучасні платформи використовують iOS Keychain і Android Keystore для безпечного зберігання даних. Управління сесіями включає автоматичний вихід після неактивності та можливість віддаленого блокування. Захист від атак забезпечується HTTPS, certificate pinning та обмеженням спроб входу [9]. GDPR і CCPA вимагають прозорості щодо збору даних і згоди користувачів [8].

Регулярні оновлення ПЗ та операційних систем усувають відомі вразливості [2]. Антивірусні програми використовують сигнатурний і поведінковий аналіз для виявлення загроз. Сегментація мережі та принцип найменших привілеїв обмежують поширення вірусів [4]. Навчання користувачів розпізнаванню соціальної інженерії знижує ризики [5].

Освіта користувачів. Навчання є ключовим для зниження ризиків через людський фактор. Користувачі повинні вміти розпізнавати фішинг, створювати надійні паролі та безпечно працювати з публічними Wi-Fi мережами. Регулярні тренінги та оновлення навчальних матеріалів підвищують обізнаність про нові загрози [5].

Ключові механізми забезпечення інформаційної безпеки з коротким описом і прикладами наведено в таблиці 2.

Таблиця 2 – Основні механізми забезпечення інформаційної безпеки

Механізм	Опис	Приклад
Шифрування	Захист даних ключем	AES, RSA, HTTPS
Автентифікація	Перевірка особи користувача	2FA, MFA, SSO
Управління доступом	Обмеження прав користувача	Keychain, Keystore
Захист від шкідливого ПЗ	Виявлення та блокування загроз	Антивіруси, оновлення
Освіта користувачів	Навчання безпечної поведінки	Тренінги з фішингу

Дослідження показало, що захист даних у мобільних застосунках вимагає комплексного підходу, який включає шифрування, автентифікацію, управління доступом, захист від шкідливого ПЗ та освіту користувачів [1]. Аналіз реальних кейсів (WhatsApp, Instagram, TikTok, Uber) підтверджує необхідність постійного вдосконалення безпеки через нові загрози, такі як атаки з використанням ШІ чи хмарних технологій [11, 12, 13]. Результати мають практичну цінність для розробників і фахівців з кібербезпеки, допомагаючи створювати надійніші системи захисту та навчальні програми. Подальші дослідження варто спрямувати на проактивний захист, зокрема методи виявлення нових видів атак і адаптацію до квантових обчислень [14].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Загрози інформаційної безпеки. Wikipedia. URL: https://uk.wikipedia.org/wiki/Загрози_інформаційної_безпеки (дата звернення: 02.01.2025).

2. Шкідливе програмне забезпечення. gov.ua. URL: https://cip.gov.ua/ua/faqs/sho-take-shkidlive-programne-zabezpechennya-yakoyi-shkodi-vono-mozhe-zavdati-komp-yuteru-chi-smartfonu (дата звернення: 03.01.2025).

3. Василенко М. Д. Шкідливі програми в контексті розуміння комп'ютерної / М. Д. Василенко, В. О. Рачук, В. М. Слатвінська // Наукові праці «Одеська юридична академія». Т. 28 /; МОН України, НУ «ОЮА». – Одеса : Видавничий дім «Гельветика», 2021. – С. 32.

4. Програмний продукт для пошуку та виявлення програм типу spyware / О. Ковальов та ін. Ukrainian Information Security Research Journal. 2022. Т. 24, № 1. С. 37. URL: https://doi.org/10.18372/2410-7840.24.16862 (дата звернення: 02.01.2025).

5. Класифікація та методи виявлення фішингових атак / Р. Штонда та ін. Кібербезпека: Освіта, Наука, Техніка. 2024. Т. 4, № 24. С. 69–80. URL: https://doi.org/10.28925/2663-4023.2024.24.6980 (дата звернення: 03.01.2025).

6. Що таке двофакторна автентифікація. gov.ua. URL: https://cip.gov.ua/ua/faqs/sho-take-dvofaktorna-avtentifikaciya-i-yak-vona-pracyuye (дата звернення: 03.01.2025).

7. Що таке атака "Людина посередині" (MITM). CyberCalm. URL: https://cybercalm.org/novyny/shho-take-ataka-man-in-the-middle-ta-yak-sebe-zahystyty/ (дата звернення: 03.01.2025).

8. Недостатній контроль доступу до API. CQR. URL: https://cqr.company/ua/web-vulnerabilities/insufficient-access-controls-for-apis/ (дата звернення: 03.01.2025).

9. Кучеренко П. В. Методи і технології захисту комп'ютерних мереж (мережний, транспортний та прикладний рівні). Телекомунікації та захист інформації. 2018. Т. 23, № 1. С. 55–56. URL: https://doi.org/10.20535/2523-4455.2018.23.1.113193 (дата звернення: 03.01.2025).

10. Ненадійні паролі – легка здобич. sfotg.gov.ua. URL: https://sfotg.gov.ua/news/1684475098/ (дата звернення: 03.01.2025).

11. Захист користувачів від кіберзагроз під час відеодзвінків Довідковий центр WhatsApp. WhatsApp. URL: https://faq.whatsapp.com/1831251587214580/?locale=uk_UA (дата звернення: 03.01.2025).

12. Троян Joker. CyberCalm. URL: https://cybercalm.org/novyny/troyan-joker-znovu-pronyk-do-magazynu-dodatkiv-na-android/ (дата звернення: 03.01.2025).

13. Атака на Facebook. Укрінформ. URL: https://www.ukrinform.ua/rubric-technology/2548069-ataka-na-facebook-hakeri-skoristalisa-nedolikom-u-sistemi-bezpeki.html (дата звернення: 03.01.2025).

14. Шаповал І. В., Лебедев Д. Ю. Алгоритм роботи пристрою AES шифратора. Проблеми інформатизації та управління. 2016. Т. 1, № 53. С. 87–91. URL: https://doi.org/10.18372/2073-4751.1.10371 (дата звернення: 03.01.2025).

15. Аналіз методів забезпечення конфіденційності даних/ С. Гнатюк та ін. Кібербезпека: освіта, наука, техніка. 2022. Т. 1, № 17. С. 167–186. URL: https://doi.org/10.28925/2663-4023.2022.17.167186 (дата звернення: 03.01.2025).

УДК 004.021

***Микитович Михайло**, здобувач ОПС фаховий молодший бакалавр*
IV курсу спеціальності «Комп'ютерні науки»
науковий керівник Кульчинська Н. З.

ДОСЛІДЖЕННЯ АЛГОРИТМІВ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ТА ЇХ ЗАСТОСУВАННЯ

Алгоритмами процедурної генерації є способи автоматичного створення даних заздалегідь заданими правилами. Вони використовуються для генерації цифрових світів, текстур, рівнів у комп'ютерних іграх, симуляціях, музиці тощо. Актуальність теми зумовлена широким використанням таких алгоритмів у сучасній розробці. Метою дослідження є аналіз основних алгоритмів процедурної генерації та прикладів їхнього практичного застосування.

Процедурна генерація як метод автоматичного створення контенту ґрунтується на двох фундаментальних підходах: стохастичних, також відомих як ймовірнісних, та детермінованих, які також ще називають визначеними, алгоритмах. Ці підходи відрізняються не лише наявністю чи відсутністю елементів випадковості, але й сферами їх оптимального застосування, обчислювальною складністю та рівнем контролю над результатом виконання.

Стохастичні алгоритми активно використовують генератори псевдовипадкових чисел для створення інших результатів при кожному виконанні. Найбільш поширеними прикладами є алгоритми на основі шумових функцій, зокрема класичний шум Перліна та його модифікація симплекс-шум. Ці методи знайшли широке застосування у генерації природних ландшафтів, де важливий ефект органічної нерегулярності. У

багатьох відеоіграх використовується комбінація шумових функцій для створення рельєфу, розподілу регіонів і розміщення печер.

Проте, навіть у стохастичних алгоритмах часто застосовують методи контролюованої випадковості, такі як seed-значення, які дозволяють відтворювати однакові результати при необхідності.

Детерміновані алгоритми характеризуються повною передбачуваністю результатів і ґрунтуються на строгих математичних правилах. Це означає, що при однакових вхідних даних результат виконання алгоритму завжди буде ідентичним, без будь-яких випадкових відхилень.

Такі алгоритми використовуються там, де потрібна точна відтворюваність або контроль над процесом генерації. Вони дозволяють будувати складні, але цілком керовані структури, наприклад, фрактальні дерева, лабіринти або розгалуження, які створюються на основі формальних граматик.

Зараз застосування процедурної генерації присутнє у різних сферах, від індустрії розваг до наукових досліджень. В ігровому дизайні процедурна генерація стала ключовим інструментом для створення масштабних віртуальних світів. Наприклад, у космічному симуляторі «No Man's Sky» комбінація процедурних алгоритмів дозволяє генерувати понад 18 квінтильйонів оригінальних планет з різноманітною флорою, фауною і рельєфом. У жанрі roguelike процедурна генерація рівнів дозволяє підтримувати довший інтерес до гри серед гравців, оскільки дозволяє грати в гру скільки завгодно, кожного разу отримуючи неповторний досвід, при цьому сучасні системи вже можуть враховувати рівень складності та логіку розміщення об'єктів.

Музична індустрія використовує процедурні методи як для створення фонових композицій, так і для експериментальних, амбієнтних творів.

Алгоритми на основі ланцюгів Маркова можуть аналізувати існуючі мелодії та генерувати нові у схожому стилі.

У 3D-моделюванні процедурні підходи революціонізували створення складних природних об'єктів. Програмні пакети на кшталт Houdini дозволяють будувати цілі міста з реалістичною архітектурою за допомогою параметричних моделей. Особливо перспективним напрямом є процедурне моделювання руйнувань і фізичних процесів, де традиційні методи потребують надмірних обчислювальних ресурсів.

Генерація текстур процедурними методами дозволяє створювати високодеталізовані матеріали з параметрично керованими властивостями. Сучасні Physically Based Rendering (PBR) текстури часто поєднують процедурні шаблони з фотографічними даними для досягнення максимального реалізму.

Найбільш інноваційним є застосування процедурної генерації в біосистемах. Еволюційні алгоритми використовуються для моделювання розвитку екосистем, генетичних мутацій і навіть для дизайну біоморфних структур у архітектурі. Дослідники MIT застосовують процедурні методи для створення штучних коралових рифів з оптимальними характеристиками.

Процедурна генерація як метод автоматичного створення контенту охоплює велику кількість спеціалізованих алгоритмів, кожен з яких має власні характеристики та сфери застосування. Ці алгоритми можна умовно класифікувати за кількома критеріями: за типом генерованих даних, наприклад ландшафти, текстури, музика тощо, за рівнем детермінованості, а також за складністю обчислювальної реалізації.

Одним з найпопулярніших алгоритмів для створення природніх текстур і ландшафтів є симплекс-шум, розроблений Кеном Перліном як вдосконалення класичного шуму Перліна. Його ключова перевага полягає в використанні

симплексних комірок замість традиційних квадратних, що значно знижує обчислювальну складність, особливо у вищих вимірах. Алгоритм найбільш ефективний для генерації хмар, лавових полів, водних поверхонь та інших природних явищ у реальному часі.

Модель Лотки-Вольтерри, розроблена в 1920-х роках, ця система диференціальних рівнянь описує динаміку двох популяцій, хижаків і жертв. Ця модель знайшла застосування для створення правдоподібних екосистем у відеоіграх та наукових симуляціях. Коливання чисельності популяції в комп'ютерній симуляції та фазовий портрет системи зображено на рисунку 1.

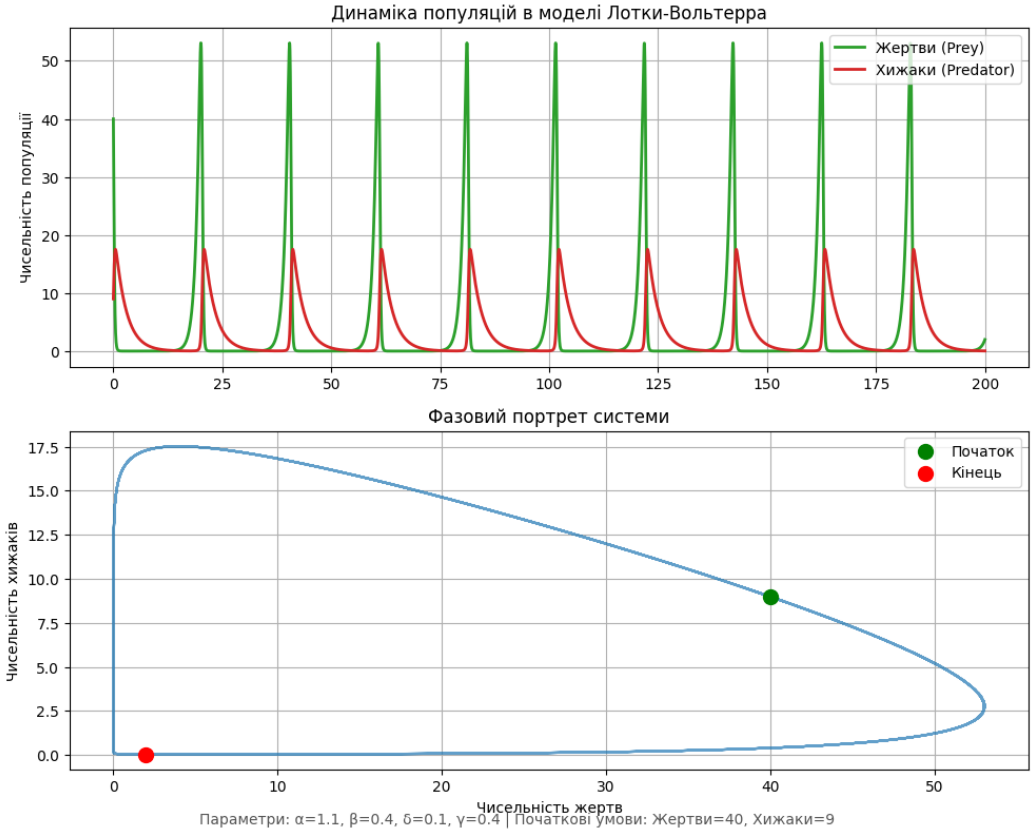

Рисунок 4 – Графік динаміки популяції

Особливістю алгоритму є його циклічність, популяції хижаків і жертв коливаються у часі, створюючи динамічну, але стабільну в довгостроковій

перспективі систему. Це дозволяє генерувати екосистеми, які еволюціонують у часі без зовнішнього втручання.

Ще одним прикладом застосування математичного моделювання в біології є L-системи, розроблені угорським біологом Аристідом Лінденмаєром у 1968 році. Ці системи, що базуються на формальних граматиках, дозволяють моделювати процеси росту біологічних організмів з вражаючою точністю. Принцип їх роботи полягає в послідовній заміні символів початкового рядка (аксіоми) згідно з набором правил переписування. Наприклад, просте правило «F → F[+F][-F]», де F позначає відрізок стебла, знаки плюс та мінус позначають кути відхилення, а квадратні дужки — розгалуження. може генерувати складні дерева з реалістичним розгалуженням. Важливою частиною L-систем є їхня масштабованість, додавання нових правил дозволяє створювати дедалі складніші структури без зміни базового алгоритму.

Алгоритми Маркова, названі на честь математика Андрія Маркова, знайшли широке застосування в генерації текстів і музики. Принцип їх роботи базується на аналізі ймовірностей переходів між станами (нотами, словами тощо). Для музичної генерації це означає створення марковської матриці, де кожен елемент визначає ймовірність переходу від однієї ноти до іншої. Сучасні реалізації часто використовують ланцюги вищого порядку, що враховують не лише поточний стан, але й попередню історію, що дозволяє досягати більш природного звучання.

Wave Function Collapse (WFC) - відносно новий алгоритм, що швидко набрав популярність серед розробників ігор. Його особливість полягає у здатності генерувати структуровані дані на основі вхідних зразків. Алгоритм працює в два етапи: спочатку аналізує вхідний зразок, виявляючи патерни та їх відноснічастоти, а потім ітеративно заповнює вихідну сітку, дотримуючись правил сумісності сусідніх елементів. Цей підхід особливо корисний для

створення архітектурних структур, міських ландшафтів та інших складних середовищ.

Алгоритм Diamond-Square, також відомий як алгоритм плазмової фрактальної поверхні, є популярним інструментом для генерації реалістичних ландшафтів. Цікавим є його поєднання детермінованого підходу, де є чіткі правила інтерполяції, з додаванням шуму на кожній ітерації. Алгоритм починає роботу з сітки заданого розміру, де спочатку значення вузлів у кутах задаються випадково або фіксуються.

На кожній ітерації він виконує дві основні операції. Перша фаза має назву «діамант», вона полягає в обчисленні середнього значення кутів кожного квадрата сітки з подальшим додаванням випадкового зсуву. Отримане значення записується в центральну точку квадрата. Друга фаза, «квадрат», передбачає аналогічні дії, але для точок, розташованих у серединах сторін сітки, де середнє значення обчислюється на основі сусідніх вузлів, і знову додається шум. Основні принципи роботи алгоритму представлено на рисунку 2 у вигляді діаграми.

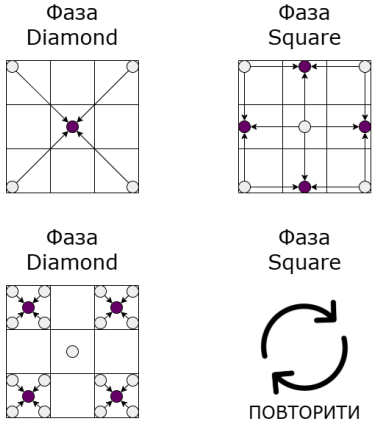

Рисунок 5 – Фази алгоритму Diamond-Square

Ці кроки повторюються для сіток зменшеного розміру, величина шуму зменшується з кожною ітерацією. Таке послідовне уточнення забезпечує

плавні переходи на великих масштабах та деталізацію на малих ділянках. В результаті формується фрактальна поверхня, яка імітує природні форми рельєфу, такі як гори, долини або пагорби.

Процедурна генерація продовжує розширювати межі своїх застосувань, поєднуючись з сучасними. Процедурна генерація дозволяє створювати величезні, різноманітні набори даних з мінімальними затратами часу та ресурсів розробників. Завдяки алгоритмічному підходу до формування інформації, ця технологія забезпечує нескінченну варіативність згенерованих компонентів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Procedural Generation. Autodesk: веб сайт. URL: https://surl.li/luqmts (дата звернення: 05.05.2025).
2. Understanding Procedural Generation: The Art of Creating Content Automatically. Monolith Academy: веб сайт. URL: https://surl.lu/unhfof (дата звернення: 10.05.2025).
3. An Introduction to Lindenmayer Systems. The University of Sussex: вебсайт. URL: https://surl.li/tyucow (дата звернення: 12.05.2025).
4. Long term behavior of solutions of the Lotka-Volterra system. Project Euclid: вебсайт. URL: https://surl.lu/iqslle (дата звернення: 13.05.2025).
5. Quantum Computing. Wave Function Collapse. IBM: вебсайт. URL: https://www.ibm.com/think/topics/quantum-computing (дата звернення: 14.05.2025).
6. The fascinating Wave Function Collapse algorithm. DEV Community: вебсайт. URL: https://surli.cc/ltfnbt (дата звернення: 14.05.2025).

УДК 004.523

***Музика Михайло**, здобувач ОПС фаховий молодший бакалавр*
IV курсу спеціальності «Комп'ютерні науки»
науковий керівник Івасьєв С.В.

РОЗРОБКА СИСТЕМИ ОБРОБКИ АУДІО У РЕАЛЬНОМУ ЧАСІ З ІНТЕГРАЦІЄЮ АУДІОЕФЕКТІВ

Реалізація гнучкої системи обробки аудіо у реальному часі вимагає ретельно продуманої архітектури, що здатна забезпечити як високу продуктивність обробки сигналів, так і зручність розширення функціональності через додавання нових ефектів. Сучасні вимоги до аудіопрограм передбачають можливість динамічного керування обробкою звуку без втрати якості та затримок, що критично важливо для інтерактивних додатків.

Мета дослідження полягає у створенні модульної системи обробки аудіосигналів у реальному часі з динамічним керуванням ефектами, реалізованої через музичний програвач на основі Qt6 та FFmpeg. Актуальність роботи обумовлена інтенсивним розвитком цифрових аудіосервісів та зростаючими вимогами до гнучких засобів звукової обробки в режимі реального часу.

Дослідження спрямоване на вивчення процесів обробки аудіосигналів у реальному часі та технологій їх програмної реалізації, при цьому основну увагу приділено архітектурним рішенням, методам та засобам створення модульної аудіосистеми з динамічним управлінням ефектами.

Наукова новизна роботи полягає у створенні гібридної архітектури реального часу, що поєднує модульну систему ефектів з динамічним

керуванням через уніфіковані інтерфейси та кросплатформну інтеграцію декодування.

На початковому етапі проєктування було розглянуто два фундаментальні підходи до інтеграції аудіоефектів: статичну компіляцію та динамічну плагінну архітектуру. Статична прив'язка ефектів до ядра системи гарантує просту інтеграцію, але кожна модифікація вимагає повного перезбирання програми, що унеможливлює швидке прототипування та залучення сторонніх розробників. Тому було прийнято рішення про використання динамічної плагінної системи, яка дозволяє відокремити реалізацію ефектів від основного коду, забезпечуючи механізм додавання нових модулів без перекомпіляції чи перезапуску додатка та значно знижуючи поріг входу для розробників.

Аналіз існуючих підходів до реалізації плагінних систем виявив три основні варіанти: використання абстрактних базових класів, фабричного патерну та прямого експорту символів. Абстрактні базові класи забезпечують строгу типізацію та уніфікацію інтерфейсів, але не надають механізму створення множинних екземплярів. Експорт символів через низькорівневі API забезпечує максимальну гнучкість, але ускладнює кросплатформну реалізацію та керування життєвим циклом об'єктів, а фабричний підхід дозволяє централізовано керувати створенням екземплярів, але може призвести до надмірної централізації логіки.

Для реалізації динамічного зв'язування було обрано рішення, що поєднує абстрактні базові класи з фабричним механізмом, дозволяючи отримати переваги обох підходів. Інтерфейс IAudioEffect визначає єдиний контракт для всіх ефектів, гарантуючи стандартизацію методів обробки звуку, а вбудований у кожен плагін фабричний механізм забезпечує створення множинних незалежних екземплярів, що критично важливо для побудови складних

ланцюгів обробки з послідовним використанням однакових типів ефектів з різними параметрами.

Фреймворк Qt6 було обрано як основу для реалізації системи завдяки його розвиненій підтримці плагінної архітектури через механізм метаоб'єктів та інтегрованим засобам для роботи з аудіо. Додатковими перевагами є кросплатформність, зріла екосистема розробки та можливість створення сучасних користувацьких інтерфейсів через декларативну мову програмування QML.

Qt6 надає вбудовану систему плагінів, яка забезпечує автоматичне виявлення і зв'язування інтерфейсів з їх реалізаціями через прозоре завантаження без значного ускладнення коду.

Процес інтеграції починається зі сканування цільових директорій для виявлення динамічних бібліотек з платформо-специфічними розширеннями (.so для Unix, .dll для Windows, .dylib для macOS). Кожна виявлена бібліотека завантажується через QPluginLoader, який інкапсулює кросплатформові особливості роботи з динамічними бібліотеками.

Ключовим етапом є верифікація сумісності плагіна через qobject_cast для безпечного приведення типів. Цей механізм перевіряє реалізацію інтерфейсу IAudioEffectFactory, повертаючи коректний вказівник при успішній верифікації або nullptr при невідповідності, що гарантує інтеграцію лише сумісних бібліотек.

Детальна діаграма послідовностей цього процесу на прикладі плагіну «Gain» зображена на рисунку 1.

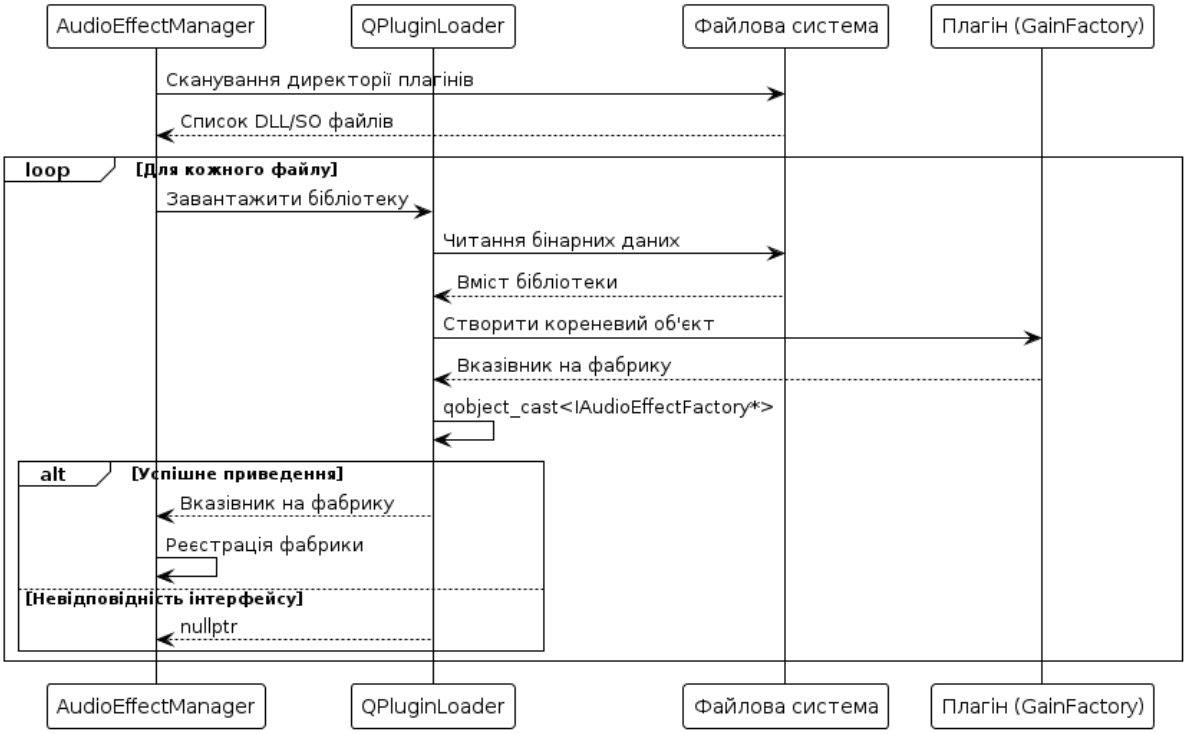

Рисунок 6 – Процес завантаження плагіну «Gain»

Клас AudioEffectManager є центральною керуючою сутністю, що відповідає за виявлення, перевірку, реєстрацію та керування аудіоефектами, реалізованими у вигляді плагінів.

Зареєстровані ефекти зберігаються у внутрішній структурі, яка поєднує фабрику плагіна, сам ефект та його стан активності. Менеджер не тільки виконує ініціалізацію, а й відповідає за подальше керування життєвим циклом ефектів: активація, деактивація, видалення та послідовне застосування активних ефектів до аудіобуфера.

Інтерфейс IAudioEffect визначає контракт для всіх аудіоефектів у системі та забезпечує уніфікований спосіб взаємодії з різними типами обробки звуку. Кожен ефект повинен реалізувати метод applyEffect для безпосереднього застосування обробки до аудіоданих, надавати власну назву через effectName

та експортувати список налаштовуваних параметрів (EffectParameter) за допомогою методу parameters.

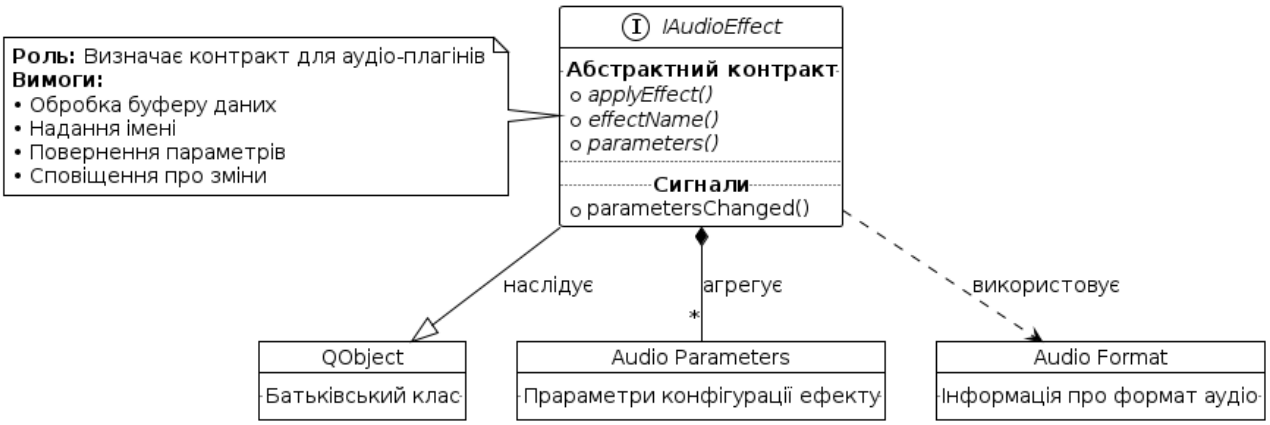

Рисунок 7 – Інтерфейс IAudioEffect

Клас EffectParameter забезпечує типобезпечне представлення налаштувань аудіоефектів та інтегрується з QML-інтерфейсом через систему властивостей Qt. Параметр характеризується іменем, поточним значенням, діапазоном допустимих значень та типом даних, який може бути: Float, Int, Bool або String. Методи класу автоматично валідують встановлювані значення відповідно до заданих обмежень та емітують сигнал valueChanged при їх зміні, що дозволяє реактивно оновлювати користувацький інтерфейс. На рисунку 3 зображено його діаграму класу.

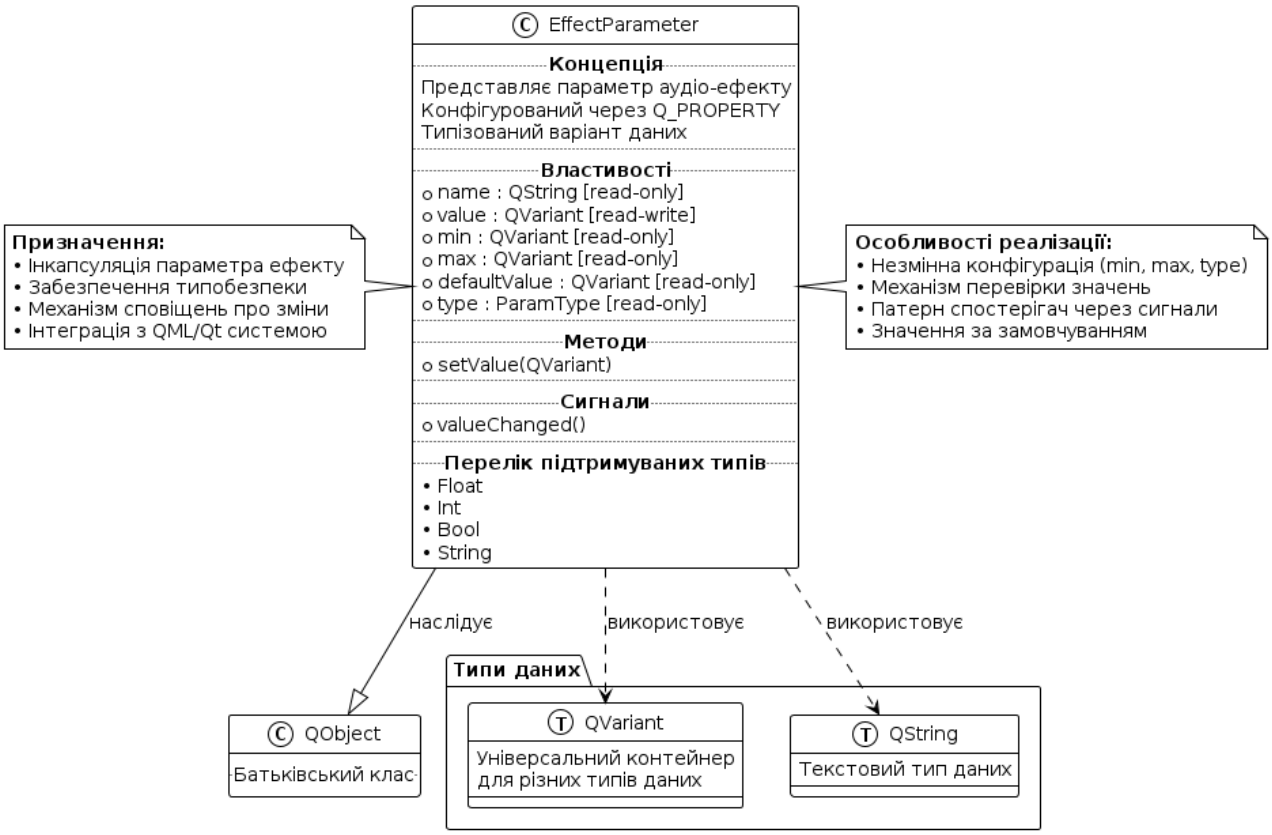

Рисунок 8 – Діаграма класу EffectParameter

Для декодування було вибрано FFmpeg через його універсальну підтримку широкого спектра аудіоформатів, високу продуктивність та надійність у роботі з мультимедійними даними. Також для інтеграції з Qt було розроблено клас FFmpegAudioDevice, який наслідується від QIODevice та інкапсулює всю логіку роботи з бібліотекою FFmpeg. Архітектура класу дозволяє використовувати його як звичайний QIODevice для QAudioSink, приховуючи складність роботи з низькорівневими API FFmpeg та забезпечуючи сигнально-слотовий механізм для повідомлення про зміни стану відтворення. Основна логіка функціонування класу FFmpegAudioDevice зображена на рисунку 4.

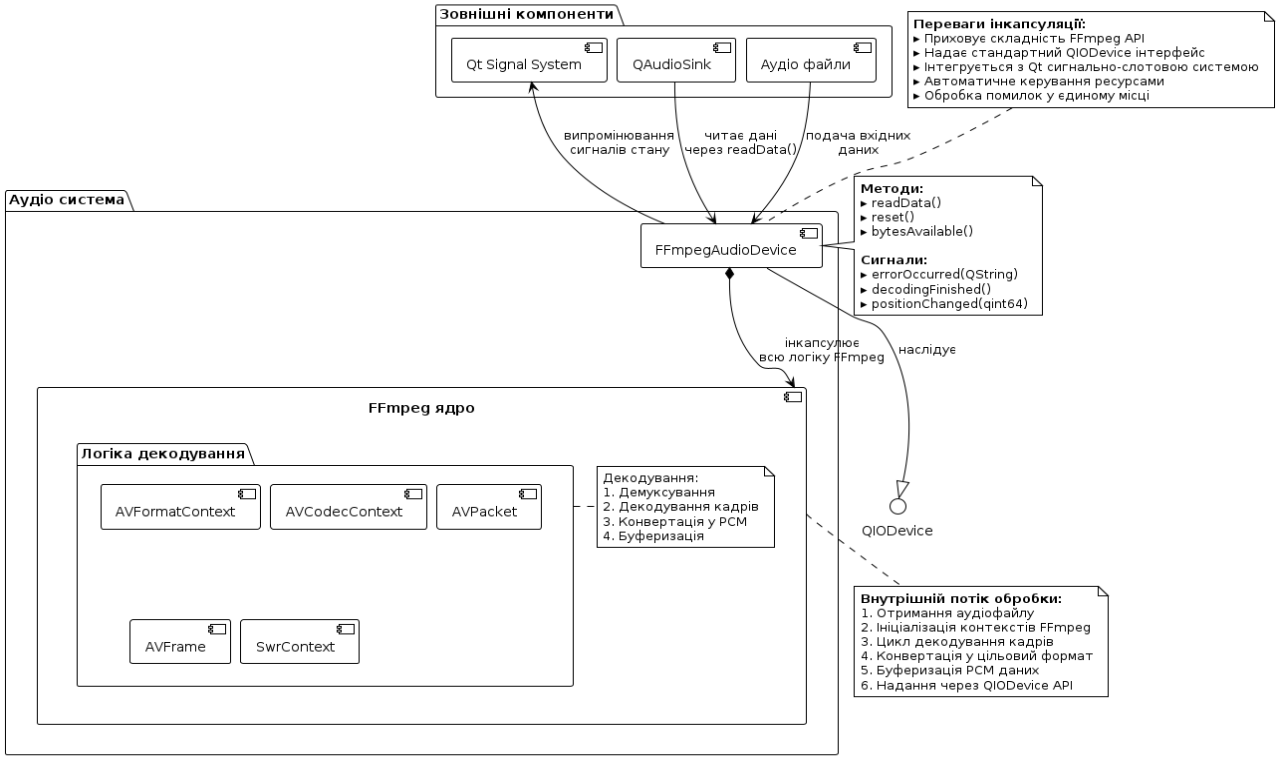

Рисунок 9 – Логіка функціонування класу FFmpegAudioDevice

Центральним компонентом системи є клас MusicPlayer, який оркеструє всією логікою обробки аудіо. Клас реалізований як singleton з інтеграцією в QML через систему властивостей Qt, що дозволяє безпосередньо використовувати його у користувацькому інтерфейсі для контролю позиції, тривалості, гучності та стану відтворення. Архітектура MusicPlayer побудована навколо взаємодії між FFmpegAudioDevice та QAudioSink (рисунок 5).

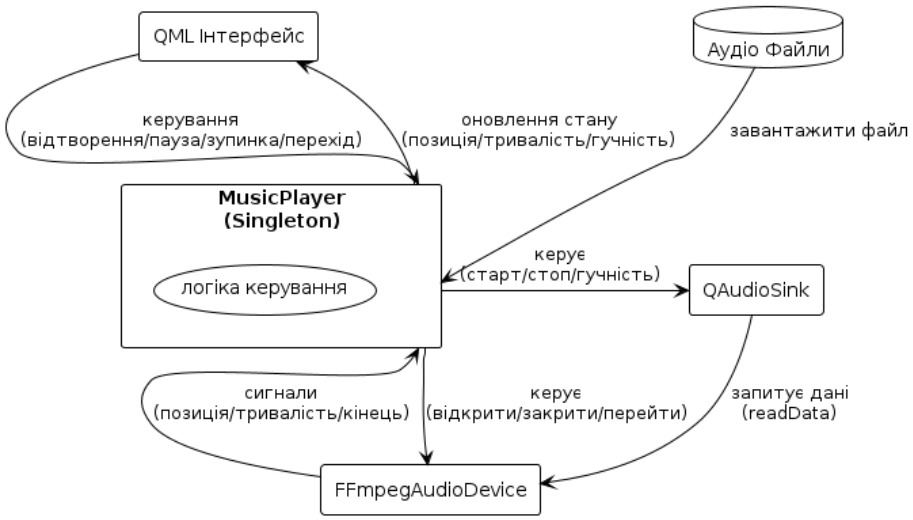

Рисунок 10 – Архітектура MusicPlayer

Для оцінки ефективності розробленої системи проведено комплексне тестування продуктивності з різною кількістю увімкнених ефектів. Тестування включало заміри часових характеристик обробки аудіосигналу в режимі реального часу при послідовному збільшенні навантаження від 0 до 4 одночасно активних ефектів.

Тестування проводилось на ноутбуці Lenovo IdeaPad Slim 5 16AHP9 з процесором AMD Ryzen 7 8845HS (8 ядер, 16 потоків, до 5.14 Ггц), інтегрованою графікою AMD Phoenix3 (AMD Radeon 780M) та 32Гб оперативної пам’яті під управлінням Arch Linux (ядро 6.14.10). Для тестування було використано 4 аудіоефекти: 16-ти смуговий еквалайзер, ефект підсилення нижніх частот, реверберація та Gain.

Підбиваючи підсумки проведеного тестування було виявлено що система демонструє відмінну продуктивність у режимі реального часу з максимальним часом обробки 185 мкс, що становить лише 0.6% від критичного ліміту буфера (30 мс). Середній час обробки складає 62-76 мкс, забезпечуючи стабільну роботу без переповнень буфера та помилок.

Збільшення кількості ефектів від 0 до 4 призводить до зростання часу обробки лише на 22%, це свідчить про ефективну архітектуру та відсутність експоненційного зростання навантаження. XY графік середніх результатів тестування зображено на рисунку 6.

Рисунок 11 – Середній час обробки

Навантаження на процесор залишається мінімальним у всіх конфігураціях: без ефектів використання CPU складає 0-0.1%, з одним ефектом зростає до 0-0.2%, з двома-трьома ефектами стабілізується на рівні 0.2-0.3% з рідкісними піками до 0.4%. Найбільший стрибок спостерігається при увімкненні чотирьох ефектів, де навантаження зростає до 1-1.2%, що пов'язано з додаванням ресурсномісткого 16-смугового еквалайзера.

Загалом система забезпечує стабільну обробку аудіо в реальному часі з великим запасом продуктивності на тестовій платформі, що робить її придатною для професійного використання з можливістю подальшого розширення функціоналу навіть на мобільних процесорах середнього класу.

У роботі було спроєктовано та розроблено універсальну систему аудіообробки на основі плагінної архітектури, що забезпечує модульну

інтеграцію ефектів через стандартизований інтерфейс. Вона оптимізована для роботи в реальному часі, підтримує гнучке конфігурування без перекомпіляції коду та забезпечує високу продуктивність, що робить її придатною для використання в реальних проєктах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. AMD Ryzen™ 7 8845HS. URL: https://www.amd.com/en/products/processors/laptop/ryzen/8000-series/amd-ryzen-7-8845hs.html (дата звернення: 08.05.2025).

2. Documentation. FFmpeg. URL: https://ffmpeg.org/documentation.html (дата звернення: 08.05.2025).

3. Factory Method. Refactoring and Design Patterns. URL: https://refactoring.guru/design-patterns/factory-method (дата звернення: 08.05.2025).

4. GeeksforGeeks. Static and Dynamic Linking in Operating Systems - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/static-and-dynamic-linking-in-operating-systems/ (дата звернення: 08.05.2025).

5. How to Create Qt Plugins | Qt 6.9. Qt Documentation | Home. URL: https://doc.qt.io/qt-6/plugins-howto.html (дата звернення: 08.05.2025).

6. PlantUML Online. PlantUML Online. URL: https://plantuml.online (date of access: 08.05.2025).

7. QAudioSink Class | Qt Multimedia | Qt 6.9.1. Qt Documentation | Home. URL: https://doc.qt.io/qt-6/qaudiosink.html (дата звернення: 08.05.2025).

УДК 004.4

***Пелех Мар'ян**, здобувач ОПС фаховий молодший бакалавр*
IV курсу спеціальності «Комп'ютерні науки»
науковий керівник Івасьєв С.В.

РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ДЛЯ ПОРІВНЯННЯ АЛГОРИТМІВ ШИФРУВАННЯ

Сучасні програми все частіше оперують великою кількістю даних, і без надійного захисту вони підпадають ризику бути отриманими небажаними сторонами. Шифрування – основний інструмент для забезпечення конфіденційності змісту, але не всі алгоритми працюють однаково: у масштабних проєктах важлива швидкість обробки, на яку сильно впливає тип алгоритму. Щоб виявити, який метод є найшвидшим і найзручнішим у роботі, був потрібен інструмент, що дозволив би на практиці порівняти алгоритми шифрування.

Основними завданнями, які були поставлені перед розробкою програмного засобу, були:

- реалізувати інтуїтивно зрозумілий графічний інтерфейс користувача з можливістю вибору алгоритму та введення вхідних параметрів;
- забезпечити функціональність тестування алгоритмів;
- забезпечити вимірювання часу операцій шифрування та розшифрування для кожного алгоритму;
- відображати результати у зручному для аналізу форматі;
- оптимізувати логіку взаємодії між криптографічними обчисленнями та візуальним інтерфейсом.

Метою цієї роботи було створення застосунку з простим інтерфейсом, який демонстрував би, скільки часу займають алгоритми шифрування RSA, AES-CBC і ChaCha20[1]. Для реалізації проєкту були використані мова QML для інтерфейсу та C++ для логічної частини програми.

Програму реалізовано із використанням криптографічної бібліотеки OpenSSL[1]. Основні функції – шифрування, розшифрування, запуск тестів – виконуються в окремому класі з точним вимірюванням часу через QElapsedTimer. Результати зберігаються у структурованому вигляді та передаються до інтерфейсу користувача за допомогою сигналів Qt[2].

Програма використовує архітектуру MVC (Model-View-Controller), як зображено на рисунку 1, що дозволяє покращити модульність проєкту, полегшує налагодження системи і забезпечує гнучке розширення функціоналу без порушення структури програми.

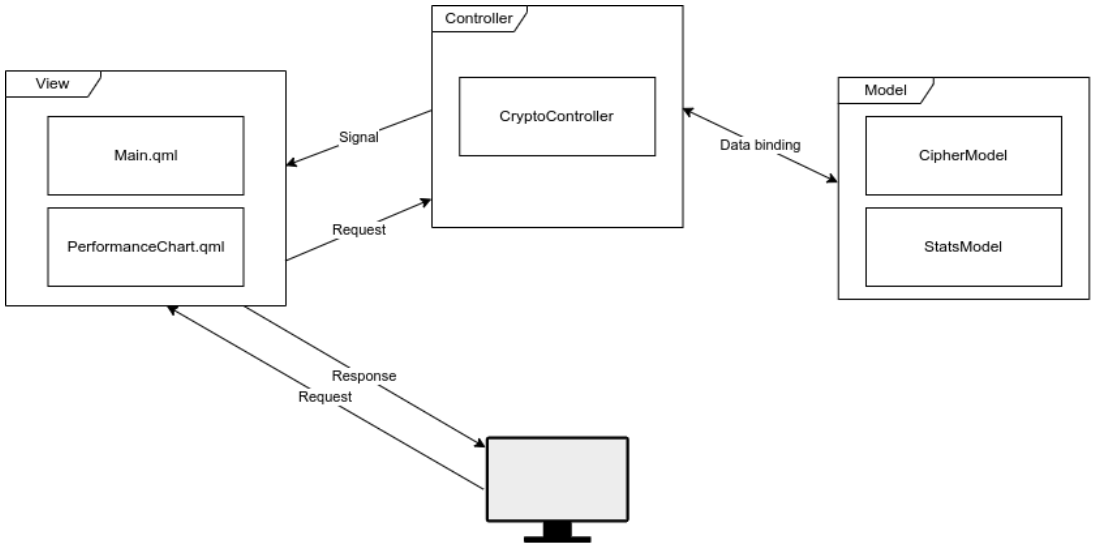

Рисунок 1 –Застосована архітектура MVC

У архітектурі MVC Model відповідає за надання певних даних і методів їх обробки; Controller відповідає за основну логіку програми, основні методи і реалізацію функціоналу; View обмінюється даними і сигналами з Controller

для встановлення взаємодії користувача з функціональною частиною застосунку через користувацький графічний інтерфейс.

За допомогою View інтерфейсу виводяться такі дані:

- шифротекст (у режимі шифрування);
- розшифрований текст (у режимі дешифрування);
- час, витрачений на кожну операцію (в мілісекундах);
- час у мілісекундах, витрачений на виконання операції шифрування для кожного алгоритму, що формує стовпчикову діаграму порівняння продуктивності для різних алгоритмів.

Це дає змогу не лише побачити результат криптографічної операції, а й оцінити її практичну ефективність на конкретному пристрої. Такий підхід допомагає зробити усвідомлений вибір алгоритму залежно від вимог до швидкодії або ресурсоємності системи, в якій він застосовуватиметься[3].

Інтерфейс створено з використанням декларативної мови QML і поділені на три відокремлені вкладки. Їх реалізовано з використанням компонентів TabBar та StackLayout, що забезпечує належне перемикання між функціональними зонами.

- Вкладка "Шифрування" (Encrypt) дозволяє користувачу ввести початковий текст (до 190 байт для RSA), а також відповідний ключ або пароль, залежно від обраного алгоритму. Після запуску операції користувач отримує шифротекст[4].

- Вкладка "Розшифрування" (Decrypt) призначена для введення шифротексту та ключа з метою розшифрування початкового повідомлення.

- Вкладка "Графік" (Graph) демонструє результати продуктивності у вигляді стовпчикової діаграми. Графік оновлюється автоматично після кожного тесту, показуючи час шифрування та розшифрування для всіх

реалізованих алгоритмів. Побудова графіка реалізована з використанням модуля Qt Charts із застосуванням BarSeries та CategoryAxis.

Крім того, ця вкладка містить кнопку "TEST ALL", натискання якої проводить автоматичне тестування усіх реалізованих алгоритмів шифрування на згенерованому рядку випадкових символів. Після завершення тестів результати миттєво відображаються у вигляді стовпчикової діаграми, що дозволяє користувачу швидко оцінити ефективність кожного алгоритму.

Інтерфейс реалізовано з мінімалістичним інтуїтивно-зрозумілим дизайном (рис. 2): чітка вертикально-орієнтована структура та звичне користувачам розміщення полів введення й кнопок робить роботу з програмою доступною для широкого спектру користувачів.

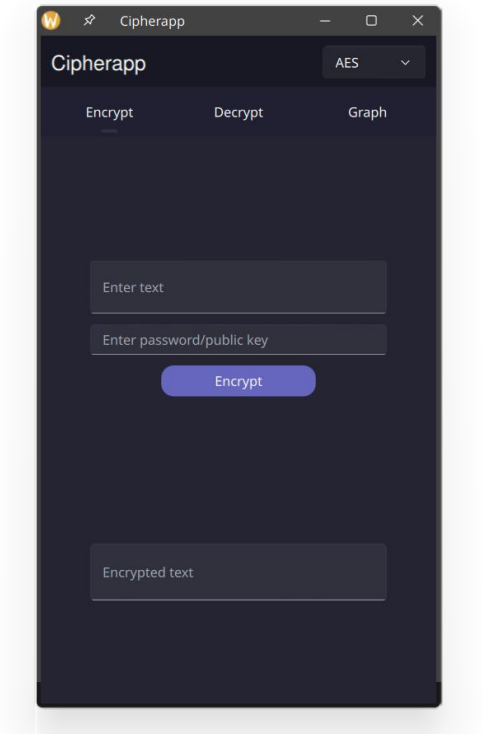

Рисунок 2 – Стартове вікно розробленого застосунку

Тестування відбувалося на пристрої з операційною системою Arch Linux, процесором Intel Core i5 1135G7 та обсягом оперативної пам’яті 16 ГБ. Для

забезпечення однакових умов порівняння усі алгоритми тестувалися на рядках довжиною 190 байт, що є максимальною довжиною вхідного тексту для RSA з ключем 2048 біт.

Результати вимірювань представлено у вигляді стовпчикової діаграми. За підсумками спостережень:

– RSA продемонстрував найнижчу продуктивність через складність обчислень, властиву асиметричному шифруванню, однак є найбільш надійним;

– AES (режим CBC) показав стабільний час шифрування та розшифрування, придатний для широкого спектра задач, хоча є менш безпечним порівняно з RSA;

– ChaCha20 забезпечив найвищу швидкість серед розглянутих алгоритмів, що обумовлено його потоковою природою та ефективною реалізацією, проте прирівнюється за надійністю до AES.

На основі отриманих результатів можна не лише порівнювати алгоритми за швидкодією, а й проєктувати комбіновані криптографічні схеми. Наприклад, доцільно використовувати RSA для захисту симетричного ключа, а сам текст шифрувати швидшим алгоритмом ChaCha20. Такий гібридний підхід поєднує високу криптостійкість асиметричних алгоритмів із продуктивністю симетричних, що є типовим для сучасних протоколів захисту інформації.

Розроблений застосунок демонструє практичну цінність як інструмент для порівняння криптографічних алгоритмів за швидкодією, а також є зручною платформою для початкового дослідження гібридних схем шифрування.

У перспективі планується:

– розширити набір підтримуваних алгоритмів, зокрема впровадити методи на основі еліптичних кривих (ECC);

– додати вимірювання іншої статистики шифрування та дешифрування;

– додати підтримку сценаріїв тестування для варіативних розмірів повідомлень;

Ці напрямки дозволять адаптувати застосунок до ширшого спектра задач і зробити його корисним у реальних дослідженнях криптографічних алгоритмів.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. OpenSSL EVP API documentation. OpenSSL : вебсайт. URL: https://www.openssl.org/docs/man3.0/man3/EVP_EncryptInit.html (дата звернення 20.05.2025).

2. Qt Charts Module. Qt Group | Documentation : вебсайт. URL: https://doc.qt.io/qt-6/qtcharts-index.html (дата звернення 20.05.2025).

3. Advanced Encryption Standard (AES). GeeksForGeeks : вебсайт. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf (дата звернення 20.05.2025).

4. What is RSA? How does an RSA work? Encryption Consulting : вебсайт. URL: https://www.encryptionconsulting.com/education-center/what-is-rsa/ (дата звернення 20.05.2025).

УДК 004

***Прохоцька Олена**, здобувач фахової*
передвищої освіти IV курсу
спеціальності «Комп'ютерна інженерія»
науковий керівник Сиротюк О.Б.

АВТОМАТИЗОВАНА СИСТЕМА МОНІТОРИНГУ ТА ОБСЛУГОВУВАННЯ РОСЛИН ІЗ ВИКОРИСТАННЯМ ІНТЕЛЕКТУАЛЬНОГО КОНТЕЙНЕРА

Система «Smart-ємність для вирощування рослин» розроблена як універсальний автоматизований пристрій для підтримки оптимальних умов росту кімнатних, тепличних та садових рослин. Вона поєднує сучасні апаратні компоненти з гнучким програмним забезпеченням і віддаленим інтерфейсом керування через Telegram. Метою створення такої системи є забезпечення стабільного мікроклімату та автоматичного поливу, що дає змогу зменшити втручання людини в процес догляду за рослинами, а також уникнути надмірного або недостатнього зволоження, перегріву чи пересушування повітря, нестачі світла або води.

Основу розробленої системи становить сучасний мікроконтролер ESP32, який відзначається високою обчислювальною потужністю, широкими комунікаційними можливостями та енергоефективністю. Він оснащений двома ядрами Tensilica LX6, що працюють на тактовій частоті до 240 МГц, а також має вбудовані модулі Wi-Fi та Bluetooth, що дозволяє реалізувати бездротову взаємодію без потреби у додаткових зовнішніх модулях. Такий вибір апаратної платформи обумовлений необхідністю забезпечення

стабільного зв’язку з користувачем, обробки даних від датчиків, керування виконавчими елементами та збереження налаштувань у постійній пам’яті.

Уся логіка керування системою побудована на багаторазовому циклі зчитування значень від підключених сенсорів, аналізу отриманих даних, прийняття рішень відповідно до заданих умов та виконання відповідних дій. Цикл опитування реалізовано у вигляді нескінченного циклу із встановленою затримкою, яка може бути змінена залежно від потреб користувача або вимог до енергоспоживання. Після кожного циклу ESP32 оновлює поточні значення параметрів, порівнює їх із допустимими межами, які задаються індивідуально для кожної обраної рослини, та приймає рішення щодо активації або деактивації системи поливу.

До ESP32 підключено набір сенсорів, кожен з яких відповідає за контроль певного параметру середовища. Основним датчиком є DHT22 – цифровий сенсор, який забезпечує одночасне вимірювання температури та відносної вологості повітря. Діапазон вимірювання температури становить від -40°C до +80°C, з точністю ±0.5°C, а діапазон вологості — від 0% до 100% із точністю ±2%. Завдяки цифровому інтерфейсу дані передаються без спотворень, що забезпечує надійність контролю мікроклімату навколо рослини.

Другим ключовим елементом є аналоговий сенсор вологості ґрунту, який дозволяє виявити рівень вологи в зоні коренів. Принцип його роботи базується на зміні електричного опору залежно від рівня вологості: чим більше води міститься в ґрунті, тим менший опір між контактами датчика. Саме показники цього сенсора є основними для прийняття рішення про запуск або зупинку поливу. Це дозволяє уникнути як надмірного зволоження, так і пересихання, які можуть бути критичними для рослини.

Ще одним важливим компонентом є сенсор рівня води в резервуарі, реалізований або як поплавковий модуль, або як ємнісний/електропровідний щуп. Завдяки цьому сенсору система може точно визначити наявність чи відсутність води. У випадку, якщо вода закінчилась, ESP32 автоматично блокує запуск поливу, а користувач отримує відповідне повідомлення у Telegram. Така функція забезпечує захист помпи від так званого «сухого ходу», який може призвести до її перегріву або поломки.

Для оцінки освітлення застосовується фоторезистор , який змінює свій опір залежно від інтенсивності світла. Цей параметр є особливо важливим для світлолюбних рослин, які потребують великої кількості світла для нормального росту та розвитку. Якщо рівень освітленості опускається нижче встановленого порогу, система може надіслати користувачеві попередження або порадити перенести рослину у краще освітлене місце.

Користувач взаємодіє із системою через Telegram-бота, інтерфейс якого розроблено таким чином, щоб бути інтуїтивно зрозумілим та доступним навіть для людей без технічної підготовки. Після запуску бота користувач бачить головне меню з актуальними показниками усіх сенсорів і переліком можливих дій. У меню можна обрати рослину для моніторингу, змінити порогові значення параметрів, увімкнути або вимкнути ручне керування поливом, а також перемикатися між автоматичним і ручним режимами.

Система підтримує два режими роботи, такі як автоматичний і ручний. У автоматичному режимі ESP32 самостійно приймає всі рішення на основі зчитаних показників. Цей режим призначений для тривалого автономного використання, наприклад, під час відпустки. У ручному режимі користувач самостійно керує запуском поливу, перевіряє стан системи, змінює параметри або переглядає журнал подій. Перемикання режимів також відбувається через меню Telegram-бота.

Для збереження налаштувань використовується файлове сховище LittleFS, яке дозволяє зберігати дані у Flash-пам’яті мікроконтролера. Усі параметри, зокрема обрана рослина, порогові межі, режим роботи та авторизовані користувачі, зберігаються у форматі JSON, що полегшує їх обробку. Завдяки цьому система після перезапуску автоматично відновлює останній стан і продовжує роботу з того місця, на якому була зупинена.

Для підвищення рівня безпеки доступ до системи обмежується авторизацією через код доступу. Після введення правильного паролю Telegram ID користувача зберігається у пам’яті, і лише він отримує повний доступ до керування системою. Усі інші запити від невідомих користувачів ігноруються, що захищає пристрій від несанкціонованого використання.

Під час етапу тестування було створено набір тест-кейсів, що охоплювали різні сценарії: перевищення або зниження порогових значень, повну відсутність води, втрату Wi-Fi-з’єднання, некоректне введення параметрів, а також спроби доступу неавторизованими користувачами. У результаті всіх тестів система продемонструвала стабільну роботу, відсутність помилок у логіці та здатність відновлення після перезавантаження або перебоїв у живленні.

Система живиться від джерела постійного струму з можливістю підключення зовнішнього блоку живлення або літієвого акумулятора. Це дозволяє використовувати її в умовах відсутності стабільного електропостачання, наприклад, у теплицях або на відкритому повітрі. У майбутньому планується реалізація енергозберігаючого режиму на базі функціоналу глибокого сну, при якому ESP32 прокидатиметься лише на короткий час для збору даних та взаємодії з ботом, що суттєво зменшить енергоспоживання пристрою.

У результаті створено функціонально завершену систему, що поєднує апаратну стабільність, зручність керування та можливість гнучкої адаптації під конкретні потреби користувача. Таким чином, «Smart-ємність для вирощування рослин» є практичним прикладом застосування концепцій Інтернету речей у побуті. Вона дозволяє автоматизувати догляд за рослинами, зберегти водні ресурси, підвищити врожайність і створити комфортні умови для росту навіть у середовищах із нестабільними кліматичними умовами. Така система може бути застосована в освітніх закладах, на балконах мешканців багатоповерхівок, у теплицях, фермерських господарствах і як частина більших агротехнічних рішень у сфері розумного сільського господарства.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Характеристики датчика температури і вологості DHT22. SparkFun Electronics: вебсайт. URL: https://www.sparkfun.com/datasheets/Sensors/Tempe (дата звернення:. 03.05.2025).

2. Технічний довідник по ESP32. Espressif Systems : веб-сайт. URL: https://www.espressif.com/sites/default/files/documentation/esp32_technical_refe (дата звернення:. 03.05.2025).

3. DHT Sensor Library / Adafruit : веб-сайт. URL: https://github.com/adafruit/DHT-sensor-library (дата звернення: 04.05.2025).

4. LittleFS for ESP32 / GitHub : веб-сайт. URL: https://github.com/lorol/arduino-esp32littlefs (дата звернення: 04.05.2025).

5. Telegram Bot API / Telegram : веб-сайт. URL: https://core.telegram.org/bots/api (дата звернення: 04.05.2025).

УДК 681.3

***Тазюк Богдан**, здобувач вищої освіти*
IV курсу спеціальності 123 «Комп'ютерна інженерія»,
керівник роботи спеціаліст вищої категорії,
викладач-методист, Павлюс В.П.

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ БІОМЕТРИЧНИХ СИСТЕМ КОНТРОЛЮ ДОСТУПУ НА БАЗІ МІКРОКОНТРОЛЕРНИХ ПЛАТФОРМ

У сучасному світі, де безпека стає дедалі важливішим аспектом як у приватному, так і в комерційному просторі, активно впроваджуються автоматизовані системи контролю доступу. Особливу нішу в цій сфері посіли біометричні технології, що дозволяють здійснювати ідентифікацію особи за унікальними фізіологічними параметрами — зокрема, відбитками пальців, розпізнаванням обличчя, райдужки ока або голосу. На тлі швидкого розвитку технологій розумного дому та Інтернету речей (IoT) біометричні системи стали ключовим елементом у створенні зручного та надійного доступу до приміщень і пристроїв.

Серед усіх біометричних методів найбільше поширення отримали системи на основі відбитків пальців. Це зумовлено низкою чинників: достатньою точністю ідентифікації, компактністю сенсорів, швидкою реакцією та відносною простотою інтеграції навіть у побутові пристрої. Порівняно з RFID-картками або PIN-кодами, біометрія значно ускладнює несанкціонований доступ, оскільки біометричні дані неможливо втратити чи передати третім особам без відома користувача.

На ринку існує безліч готових рішень — як побутового, так і професійного рівня. Наприклад, системи типу Yale Assure Lock чи Samsung

Smart Door Lock пропонують інтеграцію з мобільними застосунками та хмарними сервісами. Проте такі пристрої мають суттєві недоліки: високу вартість (часто понад 5000–10000 грн), залежність від закритих екосистем виробника, а також обмежену можливість кастомізації або локального зберігання біометричних шаблонів. Крім того, централізоване зберігання персональних даних підвищує ризики витоку або несанкціонованого доступу з боку третіх осіб.

На цьому тлі дедалі більше розробників звертають увагу на відкриті апаратні платформи, такі як Arduino, Raspberry Pi або ESP8266/ESP32. Вони дозволяють створювати повністю автономні системи контролю доступу без залежності від сторонніх серверів, із локальним зберіганням шаблонів та можливістю додаткового шифрування даних. Такий підхід відкриває широкі перспективи для індивідуального використання, освітніх проєктів, малого бізнесу та інтеграції у вже наявну інфраструктуру.

Однією з цікавих реалізацій у цьому напрямі є побудова системи контролю доступу з використанням мікроконтролера NodeMCU ESP8266, сенсора відбитків пальців DY50_MAIN_V3 та Telegram-бота для повідомлень. Основна ідея полягає у створенні мініатюрного та автономного розумного замка, який розпізнає користувача за відбитком пальця, керує замком через сервопривід, сигналізує про доступ і надсилає сповіщення власнику.

Така система є прикладом поєднання трьох важливих характеристик: по-перше, функціональність, що включає автентифікацію, логування подій та інтеграцію з месенджером; по-друге, економічна доступність — собівартість такої розробки може бути в 5–7 разів нижчою за комерційні аналоги; по-третє, відкритість, що дозволяє гнучке налаштування та розвиток системи під конкретні потреби користувача.

Стабільна робота такої конструкції забезпечується завдяки використанню бібліотек Arduino, реалізації HTTPS-з'єднань із Telegram API, а також оптимізованому алгоритму обробки подій. Система може працювати автономно впродовж тривалого часу, має невелике енергоспоживання та здатна працювати в широкому діапазоні температур.

Що особливо важливо — уся біометрична інформація обробляється локально на пристрої, а отже, не передається до сторонніх серверів. Це істотно підвищує рівень конфіденційності та безпеки, що в сучасних умовах є вирішальним фактором при виборі технологій контролю доступу.

Загалом, використання відкритих мікроконтролерних платформ для побудови біометричних систем контролю доступу є перспективним напрямом як з технічної, так і з соціальної точки зору. Воно дозволяє забезпечити безпеку без компромісу між приватністю, вартістю та зручністю. Такі рішення мають потенціал для подальшого розвитку — зокрема, через інтеграцію з мобільними додатками, хмарними сховищами (за бажанням користувача), підтримкою NFC або розпізнаванням обличчя як додатковим фактором автентифікації.

Проте навіть у сучасних реалізаціях таких систем усе ще зберігаються суттєві обмеження. Це, зокрема, висока вартість комерційних пристроїв, обмеження в налаштуванні, відсутність повного контролю над даними користувачів, а також необхідність постійного підключення до хмарних сервісів. Для багатьох користувачів ці чинники є критичними, особливо в умовах, коли приватність і автономність мають першочергове значення.

Враховуючи ці недоліки, було вирішено створити альтернативну систему біометричного контролю доступу з акцентом на відкритість, доступність, безпеку та простоту впровадження. Така система мала відповідати низці вимог: працювати автономно без підключення до сторонніх

серверів, зберігати біометричні дані виключно локально, бути сумісною з месенджером Telegram для сповіщення, а також бути фінансово доступною для масового користувача.

Результатом розробки стала система, побудована на базі мікроконтролера ESP8266, яка включає такі компоненти:

– Біометричний сенсор DY50_MAIN_V3. Він відповідає за зчитування та автентифікацію користувача за відбитками пальців.

– Сервопривід MG90S. Виконує фізичне розблокування замка при успішному розпізнаванні.

– Світлодіоди та активний бузер. Служать для візуальної та звукової індикації процесів доступу.

– Інтеграція з Telegram через API. Дозволяє надсилати повідомлення власнику про результат спроби доступу.

– Захищене з’єднання HTTPS (TLS 1.2). Гарантує безпечну передачу даних між пристроєм і Telegram.

Особливістю розробленої системи є відмова від хмарної інфраструктури та повна автономність, що дозволяє її впровадження у приватних будинках, офісах або невеликих організаціях. Крім того, невисока вартість апаратних компонентів робить таку систему конкурентною навіть на тлі масових рішень, а відкритий код та архітектура дозволяють адаптувати її до різних сценаріїв використання.

Поява подібних рішень у сфері освіти, малих підприємств і домашнього використання сприятиме популяризації ідей технічної самостійності, цифрової безпеки та контролю над власними даними. Це ще раз доводить, що відкриті технології у поєднанні з біометрією — не просто альтернатива, а цілком конкурентоспроможна реальність.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ESP8266 Wi-Fi Module Overview. Espressif Systems. Вебсайт. URL: https://www.espressif.com/en/products/modules/esp8266 (дата звернення: 22.05.2025).

2. Fingerprint Sensor DY50_MAIN_V3 Datasheet. Digital Biometrics Inc. Вебсайт. URL: https://digitalbiometrics.com/dy50_main_v3-datasheet (дата звернення: 22.05.2025).

3. Telegram Bot API Documentation. Telegram. Вебсайт. URL: https://core.telegram.org/bots/api (дата звернення: 22.05.2025).

4. ArduinoJson Library. Arduino. Вебсайт. URL: https://arduinojson.org/ (дата звернення: 22.05.2025).

5. Secure Communication with TLS on ESP8266. Espressif Forum. Вебсайт. URL: https://esp8266.net/secure-communication-tls/ (дата звернення: 22.05.2025).

6. Home Automation and Biometric Security Systems Review. IEEE Access. Вебсайт. URL: https://ieeexplore.ieee.org/document/9043715 (дата звернення: 22.05.2025).

УДК 341.64

Чичота Яна, *здобувач фахової перед вищої освіти*
IV курсу спеціальності «Комп'ютерна інженерія»,
науковий керівник Сиротюк О.Б.

РОЗУМНА РУКАВИЧКА ДЛЯ ПІДВИЩЕННЯ БЕЗПЕКИ ТА АВТОНОМНОСТІ ЛЮДЕЙ ІЗ ВАДАМИ ЗОРУ

У сучасному світі питання забезпечення доступності й інклюзії набуває дедалі більшої актуальності, особливо коли мова йде про створення

комфортних умов для безпечного й самостійного пересування людей із вадами зору. Обмежене сприйняття навколишнього простору через відсутність або суттєве погіршення зору ускладнює орієнтацію та мобільність, що нерідко стає причиною травм, втрати впевненості у власних силах і навіть соціальної ізоляції. У зв’язку з цим постає важливе завдання пошуку та розробки сучасних технологічних рішень, здатних компенсувати нестачу зорової інформації за рахунок сенсорних сигналів, що завчасно повідомляють про зміну навколишнього середовища.

Сенсорні технології швидко розвиваються паралельно з новими тенденціями в Інтернеті речей (IoT) та вбудованих системах, багато вдосконалень були розроблені в сучасних допоміжних пристроях [1]. Інтеграція кількох датчиків на єдиній платформі для подолання обмежень окремих датчиків стає все більш популярною. Також зростає кількість досліджень допоміжних пристроїв на основі мікроконтролерів, які дозволяють швидше сповіщати користувача за допомогою різних типів виконавчих механізмів та бездротових зворотних зв’язків [2]. Наприклад, запропоновано кермування міні-роботом на колесах, прикріпленим до білої тростини, для забезпечення вібротактильного зворотного зв’язку з відчуттям напрямку[3].

Незважаючи на технологічну революцію, допоміжні пристрої не були успішно прийняті та використані великою кількістю людей. Основні проблеми, пов’язані з обмеженим використанням цих розумних пристроїв, включаючи високу ціну, безпеку, орієнтацію, швидкість, мобільність, портативність та оптимізацію методів [3]. У деяких випадках багато варіацій сповіщень для користувача можуть бути заплутаними та менш інтуїтивно зрозумілими, і користувачі зазвичай віддають перевагу отриманню спрощених

сигналів без необхідності обробляти багато необроблених даних зі зворотних зв’язків.

З огляду на ці потреби постає завдання розробки спеціального пристрою, що забезпечуватиме точне й надійне виявлення перешкод у просторі та інформування користувача у зрозумілій формі. Такий прилад повинен підвищувати мобільність і безпеку людей із вадами зору, сприяти розширенню їхньої незалежності та покращувати загальну якість життя. Процес розробки подібного приладу охоплює як створення апаратної платформи, так і розробку програмного забезпечення для реалізації функцій вимірювання дистанції, формування звукових повідомлень та надсилання сигналу тривоги.

«Розумна рукавичка» – компактний пристрій, що поєднує функції виявлення об’єктів у навколишньому середовищі й звукового сповіщення, а також можливість надсилати координати користувача на мобільний телефон через Bluetooth [4]. Така система призначена для того, щоби забезпечувати своєчасне попередження про потенційні небезпеки, супроводжувати користувача звуковими сигналами й гарантувати можливість виклику допомоги у надзвичайній ситуації.

Принцип роботи пристрою ґрунтується на безперервному моніторингу навколишнього простору лазерним дальноміром VL53L1X [5]. При наближенні об’єкта до руки користувача мікроконтролер ESP32 [6] миттєво надсилає команду аудіомодулю JR6001 [7], який відтворює відповідне голосове повідомлення про появу перешкоди.

Додатковою важливою функцією пристрою є тривожна кнопка, при натисканні якої миттєво формується повідомлення про надзвичайну ситуацію. Отримані на смартфоні координати передаються на заздалегідь заданий номер, що дозволяє швидко сповістити відповідальних осіб. Підтвердження

відправлення сигналу супроводжується голосовим повідомленням, яке відтворюється аудіомодулем для спокою й інформованості користувача.

Усі ці можливості реалізуються у компактній архітектурі навколо мікроконтролера ESP32, який здійснює управління сенсором, аудіомодулем, Bluetooth й тривожною кнопкою. Мікроконтролер забезпечує обробку вхідних сигналів у реальному часі, своєчасне інформування користувача й обмін даними зі смартфоном [6]. Конструктивна простота й автономність системи роблять її придатною для повсякденного використання.

Коли пристрій вмикається, мікроконтролер ESP32 проводить ініціалізацію всіх підключених периферійних модулів. Він налаштовує інтерфейс I2C для зв'язку з датчиком VL53L1X, встановлює UART-з'єднання з аудіоплеєром JR6001 та активує вбудований Bluetooth-модуль, переходячи в режим очікування для підключення до смартфона. На цьому етапі також виконуються внутрішні перевірки працездатності пам'яті та периферії. Після успішної ініціалізації ESP32 переходить у постійний режим моніторингу навколишнього середовища та очікування команд.

У нормальному режимі роботи сенсор VL53L1X, використовуючи технологію Time-of-Flight, безперервно вимірює відстань до об'єктів у своєму полі зору та передає ці дані до мікроконтролера ESP32. ESP32 в реальному часі аналізує отримані показники, порівнюючи виміряні відстані з динамічно встановленими порогами безпеки. Якщо виміряна відстань до будь-якого об'єкта стає критично малою, сигналізуючи про потенційну перешкоду мікроконтролер надсилає команду модулю аудіоплеєра JR6001. JR6001, у свою чергу, відтворює через підключений динамік заздалегідь записане голосове повідомлення, що інформує користувача про наявність перешкоди («Попереду перешкода», «Підвищення" або «Пониження»). Чим ближче об'єкт, тим інтенсивнішим або частішим може бути звуковий сигнал, надаючи

інтуїтивну інформацію про ступінь небезпеки. Після того, як перешкода зникає з поля зору датчика або користувач віддаляється на безпечну відстань, звукове сповіщення автоматично припиняється.

У разі виникнення небезпечної ситуації, користувач може активувати протокол екстреного сповіщення, натиснувши на "кнопку тривоги". Після натискання кнопки, мікроконтролер ESP32 негайно передає координати взяті зі смартфона і надсилає його на смартфон, який використовується опікуном. Паралельно модуль аудіоплеєра JR6001 відтворює голосове підтвердження відправлення сигналу тривоги, заспокоюючи користувача.

У результаті проведеної розробки спроектовано функціональний пристрій для підвищення безпеки й мобільності людей із вадами зору. Він простий у використанні, енергоефективний і надійний; забезпечує своєчасне виявлення перешкод і подає зрозумілі для користувача голосові повідомлення, що дозволяють краще орієнтуватися в просторі й уникати травматичних ситуацій. Можливість надсилання сигналу тривоги й координат при натисканні кнопки розширює функціонал приладу й робить його корисним у критичних обставинах. Архітектура, побудована на базі мікроконтролера ESP32, гарантує автономність, швидкодію й низьке енергоспоживання системи, а використані сенсори й модулі відповідають сучасним технічним вимогам. Запропонована розробка враховує потреби користувачів, не потребує складного навчання й може стати важливим засобом підвищення якості життя осіб із вадами зору. Отримані результати свідчать про доцільність використання таких рішень для розвитку доступного й безпечного середовища та відкривають перспективи подальшого вдосконалення й розширення функціоналу пристрою.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. New device could help visually impaired avoid obstacles, research suggests. The Guardian. 2021. 22 Jul. URL: https://www.theguardian.com/society/2021/jul/22/new-device-could-help-visually-impaired-avoid-obstacles-research-suggests?utm_source=chatgpt.com (дата звернення: 04.05.2025).

2. Wafa Elmannai and Khaled Elleithy. Sensor-based assistive devices for visually-impaired people: Current status, challenges, and future directions. *Sensors*. 2017. № 17(3). p. 565 URL: https://www.mdpi.com/1424-8220/17/3/565 (дата звернення: 04.05.2025).

3. N.S. Ahmad, N.L. Boon and P. Goh, Multi-Sensor Obstacle Detection System Via Model-Based State-Feedback Control in Smart Cane Design for the Visually Challenged. *IEEE Access*. Vol. 6, 29 October 2018. pp. 64182-64192. URL:https://www.researchgate.net/publication/328605992_Multi Sensor_Obstacle_Detection_System_Via_Model-Based_State-Feedback_ Control_in_Smart_Cane_Design_for_the_Visually_Challenged (дата звернення: 04.05.2025).

4. Mr. Venkat Ghodke, Miss. Pragati Jain, Miss. Tanishqa Bhalekar, Mr. Yash Dhamdhere. Third Eye for Blind Using Ultrasonic Sensor *Tuijin Jishu/Journal of Propulsion Technology.* Vol. 45. №3 (2024) URL: https://www.propulsiontechjournal.com/index.php/journal/article/view/7309/4720 (дата звернення: 04.05.2025).

5. VL53L1X Time-of-Flight Distance Sensor — [Офіційна документація Adafruit]. URL: https://learn.adafruit.com/adafruit-vl53l1x (дата звернення: 02.05.2025).

6. Посібник з мікроконтролерів ESP32. Глобальний дистриб'ютор електронних компонентів ARIAT TECHNOLOGY LIMITED. URL: https://ua.ariat-tech.com/blog/ESP32-Microcontroller-Guide.html (дата звернення: 02.05.2025).

7. Instructables. JR6001 MP3 Module Controlled by Arduino. Instructables. URL: https://www.instructables.com/JR6001-MP3-Module-Controlled-by-Arduino/ (date of access: 22.05.2025).

УДК 004.418

Щотчук Роман*, здобувач фахової передвищої освіти IV курсу спеціальності «Комп'ютерні науки» науковий керівник Чубей О. О.*

РОЗРОБКА ТА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКУ: «МОБІЛЬНИЙ ТРЕНАЖЕР З ОСНОВ ГРИ НА БАЯНІ»

Сучасний світ характеризується стрімким розвитком цифрових технологій, що трансформують освітню сферу. Музична освіта, зокрема навчання грі на баяні, є складним процесом, що потребує значної координації та технічної вправності. При цьому ринок мобільних додатків пропонує обмежений вибір якісних спеціалізованих інструментів для навчання грі на цьому народному інструменті. Ця обставина обґрунтовує високу актуальність розробки мобільного тренажера "Bayan Trainer", який здатен заповнити існуючу прогалину та зробити навчання доступнішим.

Перед розробкою було проаналізовано існуючі рішення. Такі додатки, як Accordion Chromatic Cassoto, пропонують реалістичну симуляцію гри, але не містять навчальних матеріалів для початківців. Платформа Piano by Yousician, використовує ефективні інтерактивні та гейміфіковані методики навчання, але не адаптовані під специфіку баяна, зокрема особливості лівої клавіатури та керування міхом. Інші застосунки, як Accordion Piano, хоч і мають вбудовані уроки, проте їхній функціонал обмежений у безкоштовній версії через рекламу та преміум-доступ. Аналіз підтвердив необхідність створення комплексного рішення, орієнтованого саме на баяністів.

Розробка додатку велася з використанням мови програмування Dart та кросплатформенного фреймворку Flutter. Було спроєктовано модульну функціональну структуру, що логічно розділяє можливості системи.

Відеоуроки в модулі "Практика"

Важливою складовою навчального процесу є інтеграція відеоматеріалів, що робить навчання більш наочним та ефективним. За прикладом провідних освітніх платформ, як-от Yousician, які активно використовують відео для пояснення складних моментів, у додатку "Bayan Trainer" реалізовано модуль "Практика". Цей розділ містить колекцію авторських відеоуроків, які демонструють техніку виконання вправ та пісень. Завдяки візуальному демонструванню та детальним поясненням, користувачі можуть значно легше засвоювати нові технічні прийоми та музичну теорію. У майбутньому планується постійне розширення бібліотеки уроків та пісень, що забезпечить довготривалу актуальність та цінність додатку для користувачів.

Створено ключові моделі для представлення сутностей, такі як Note (музична нота), Exercise (вправа) та UserProgress (прогрес користувача).

Керування станом: Для ефективного оновлення інтерфейсу та керування даними застосовано патерн Provider, що дозволило відокремити бізнес-логіку від представлення.

Для збереження налаштувань користувача, прогресу та іншої інформації між сесіями використовується плагін shared_preferences, що реалізує сховище за принципом "ключ-значення". Складні структури даних перед збереженням серіалізуються у формат JSON.

Архітектура мобільного додатку "Bayan Trainer" спроєктована з урахуванням необхідності подальшої підтримки та розширення навчальної бази. Функціональна структура системи передбачає наявність окремого адміністративного модуля, призначеного для розробника або адміністратора. Цей модуль надає інструменти для додавання та редагування навчального контенту, керування списком вправ та їх рівнем складності, а також для перегляду статистики користувачів. Такий підхід дозволяє підтримувати актуальність освітніх матеріалів та гнучко реагувати на потреби аудиторії без необхідності випускати оновлення самого додатку. Це забезпечує довготривалий життєвий цикл продукту та його постійне вдосконалення.

Додаток "Bayan Trainer" реалізує наступні ключові модулі:

– модуль автентифікації: забезпечує можливість реєстрації, входу та виходу користувача, що є основою для персоналізації навчання та збереження прогресу;

– навчальний модуль: надає доступ до теоретичних матеріалів, що охоплюють будову баяна, нотну грамоту та правильну постановку рук, доповнених візуальними ілюстраціями;

– інтерактивний тренажер: включає віртуальну імітацію правої (мелодійної) та лівої (акордової) клавіатур баяна. Відтворення звуків

реалізовано за допомогою плагіна audioplayers, що забезпечує низьку затримку для реалістичного відчуття гри;

– бібліотека творів: містить список пісень та вправ, для кожної з яких можна переглянути ноти та прослухати аудіо запис;

– модуль прогресу та календаря: дозволяє користувачам відстежувати свої досягнення, відзначати дні занять у календарі та переглядати статистику навчання;

– система налаштувань: надає можливість змінювати тему інтерфейсу (світла/темна), регулювати гучність звуків та обирати мову.

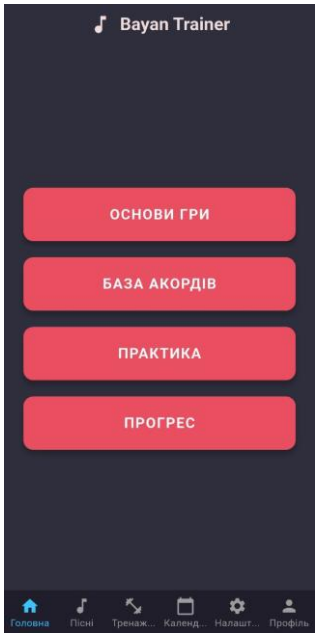

Рисунок 1 – Головна сторінка додатку

У результаті виконання роботи було розроблено функціональний мобільний додаток, що відповідає поставленим цілям. Проєкт демонструє успішне застосування сучасних технологій кросплатформенної розробки для створення спеціалізованого освітнього продукту. Додаток вирішує проблему обмеженої кількості якісних інструментів для самостійного навчання грі на

баяні, сприяє популяризації народного інструменту та має значний потенціал для подальшого розвитку та комерціалізації. Подальші перспективи включають розширення бібліотеки уроків, впровадження нових інтерактивних функцій та інтеграцію з соціальними мережами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна документація фреймворку Flutter. URL: https://flutter.dev/ (дата звернення: 05.05.2024 р.)
2. Репозиторій пакетів для Flutter та Dart. URL: https://pub.dev/ (дата звернення: 10.05.2024 р.)
3. Освітня музична платформа Yousician (для аналізу аналогів). URL: https://yousician.com/ (дата звернення: 20.02.2024 р.)
4. Мобільний додаток Accordion Piano (для аналізу аналогів). URL: https://play.google.com/store/apps/details?id=com.bilkon.accordion (дата звернення: 20.02.2024 р.)
5. Набір звукових семплів фортепіано у форматі MP3. URL: https://github.com/fuhton/piano-mp3 (дата звернення: 15.05.2024 р.)
6. Безкоштовні звукові ефекти Pixabay Sound Effects. URL: https://pixabay.com/sound-effects/ (дата звернення: 16.05.2024 р.)
7. Документація пакета audioplayers для відтворення звуку. URL: https://pub.dev/packages/audioplayers (дата звернення: 12.05.2024 р.)
8. Документація пакета provider для керування станом. URL: https://pub.dev/packages/provider (дата звернення: 11.05.2024 р.)